Chapter 10

{ Empty }

Chapter 11

Curve Fitting

11.1 Polynomial Least-Squares Curve Fit

A. Purpose

This subroutine determines a polynomial that fits a given discrete set of data in the sense of minimizing the weighted sum of squares of residuals. The fitted polynomial can be constrained to match some data points exactly by appropriate setting of the a priori standard deviations of the data errors.

Auxiliary subroutines, described in Chapter 11.2, may be used to evaluate, differentiate, or integrate a polynomial produced by this fitting procedure.

B. Usage

Program Prototype, Single Precision

```
INTEGER M, NMAX, NDEG
REAL    X(≥ M), Y(≥ M), SD(≥ M or = 1), P(≥ NMAX+3), SIGFAC, W((NMAX+3)
        * (NMAX+3))
LOGICAL SEEKN, COMTRN, CHBBAS
```

Assign values to M, X( ), Y( ), SD( ), NMAX, SEEKN, COMTRN and CHBBAS. If COMTRN = .FALSE. values must also be assigned to P(1) and P(2).

```
CALL SPFIT (M, X, Y, SD, NMAX, SEEKN, COMTRN, CHBBAS, P, NDEG, SIGFAC, W)
```

Following the CALL to SPFIT, one may wish to use SCPVAL or SMPVAL to evaluate the fitted polynomial at specific points, SCPDRV or SMPDRV to differentiate the polynomial, or SCPINT or SMPINT to integrate the polynomial. See Chapter 11.2 for descriptions of these subprograms.

Argument Definitions

M [in] Number of data points.

X( ) [in] Array of values of the independent variable. Values need not be ordered and may be repeated.

Y( ) [in] Array of values of the dependent variable indexed to correspond to the X( ) values.

SD( ) [in] If SD(1) > 0., each SD(I) must be positive and will be assumed to be the user's a priori estimate of the standard deviation of the uncertainty (observational errors, etc.) in the corresponding data value Y(I).

If SD(1) < 0., |SD(1)| will be used as the a priori standard deviation of each data value Y(I). In this case the array SD( ) can be dimensioned SD(1) since locations following SD(1) will not be used.

NMAX [in] Specifies the maximum degree polynomial to be considered. Require NMAX ≥ 0. If NMAX > M-1 the subroutine will function as though NMAX = M-1.

SEEKN [in] If SEEKN = .TRUE. the subroutine will determine the optimum value of NDEG ≤ NMAX in the sense described below in Section D.

If SEEKN = .FALSE., the subroutine will set NDEG = NMAX unless this is a singular or nearly singular problem, in which case NDEG will be reduced as necessary.

COMTRN[in] If COMTRN = .TRUE., the subroutine will determine the transformation parameters P(1) and P(2) as described in Section D.

If COMTRN = .FALSE., the subroutine will use the values of P(1) and P(2) that have been set by the user.

CHBBAS [in] If CHBBAS = .TRUE., the subroutine will use the Chebyshev basis. If CHBBAS = .FALSE., the subroutine will use the monomial basis. The Chebyshev basis is recommended for better numerical stability, particularly for NMAX greater than about six.

P( ) [inout] Parameter vector defining a polynomial according to Eqs. (1) and (2) or (3) and (4). Values of P(1) and P(2) may either be set by the user, or by the subroutine, depending upon the value of COMTRN. Values of P(J+3), J=0, ..., NMAX will be computed by the subroutine. If NDEG < NMAX, the subroutine will set P(I+3)=0, for I=NDEG+1, ..., NMAX.

NDEG [out] Set by the subroutine to indicate the degree of the fitted polynomial. On return NDEG will satisfy $0 \le NDEG \le NMAX$ if the computation was successful and will be set to -1 if an error occurred.

SIGFAC [out] Number computed by the subroutine. See discussion of SIGFAC in Sections C and D following.

W( ) [scratch] Array of working storage.

Modifications for Double Precision

For double precision usage change the REAL type statements to DOUBLE PRECISION and change the subroutine name from SPFIT to DPFIT. Change the names of the auxiliary subroutines mentioned above to DCPVAL, DMPVAL, DCPDRV, DMPDRV, DCPINT, and DMPINT, respectively.

C. Examples and Remarks

Example: Given a set of 12 data pairs (x_i, y_i), compute the weighted least squares polynomial fit to these data letting the subroutine determine the preferred polynomial degree, not to exceed 8. After computing the least squares polynomial p(x), compute and tabulate the quantities x_i, y_i, $p(x_i)$, and $r_i = y_i - p(x_i)$.

The program DRSPFIT carries out this computation using subroutines SPFIT and SCPVAL. The output is shown in ODSPFIT. Note that the value of NDEG selected by the subroutine is 7.

Transformation of the Independent Variable

For the best numerical accuracy, both in determining the best fitting polynomial and in the later evaluation of the polynomial, it is generally advisable to use transformation parameters P(1) and P(2) causing the transformed variable u of Eqs. (1) or (3) to range from -1 to +1. This condition can be obtained by setting COMTRN = .TRUE.

When fitting with polynomials of degree higher than about six, the Chebyshev basis generally gives better numerical accuracy than the monomial basis. To obtain the potential numerical advantages of the Chebyshev basis it is essential to cause the transformed variable, u, to range from -1 to +1.

To force an identity transformation (i.e., effectively no transformation) of the independent variable, set COMTRN = .FALSE., P(1) = 0., and P(2) = 1.

Interpretation of SIGFAC

If the user has set all SD(I) = 1.0, or equivalently, set SD(1) = -1.0, SIGFAC can be interpreted as an a posteriori estimate of the standard deviation of the error in each Y(I) value. More generally, whatever values the user has assigned to the SD(I)'s, the a posteriori estimate of the standard deviation of the error in Y(I) is SIGFAC*SD(I).

Equality Constraints

If the user wishes the fitted polynomial to agree to machine accuracy with one or more of the data points, this can be accomplished by setting the SD() value for such points much smaller than for the other points.

Let ε denote the machine precision, i.e., ε = R1MACH(3) in S.P. or ε = D1MACH(3) in D.P. For some value of c in the range $0.5 \le c \le 0.75$, we suggest setting SD(I) = ε^c for the points at which an exact fit is desired and SD(I) = 1.0 for all other points. For example if $\varepsilon = 10^{-8}$ set SD(I) in the range 10^{-4} through 10^{-6} for the exact fit points.

Using $c < 0.5$ may not produce sufficiently small residuals at the desired points of exact fit, whereas setting $c > 0.75$ may trip the near-singularity test in these subroutines leading to an unwanted alteration of the problem.

Note that it is not mathematically reasonable to attempt to force exact fits at more than NMAX+1 points.

D. Functional Description

The user provides data (x_i, y_i, s_i), for i=1, ..., M, and an integer NMAX. If SEEKN = .FALSE. and the problem is not rank-deficient or extremely ill-conditioned, the subroutine simply determines the polynomial $p_n(x)$ of degree n=NMAX that minimizes the following weighted sum of squares of residuals:

$$p_n^2 = \sum_{i=1}^{M} \left[\frac{y_i - p_n(x_i)}{s_i} \right]^2 .$$

The subroutine also computes

$$\sigma_n = \left[\frac{p_n^2}{\max(1, M-n-1)} \right]^{1/2}$$

and returns this value as SIGFAC.

The subroutine makes use of subroutines SROTMG and SROTM (or DROTMG and DROTM) from the BLAS to implement a sequential Modified Givens orthogonal transformation method of reducing the matrix of the least squares problem to triangular form (Ref. 1), and then solves the triangular system. This method has excellent numerical stability.

After triangularizing the matrix of the problem for degree NMAX, the quantities p^2_n and σ^2_n may be computed inexpensively for all degrees n from zero through NMAX. Thus, if the user sets SEEKN = .TRUE. the subroutine computes these values of p^2_n and computes

$$\text{CMIN} = \min \{\sigma^2_n : 0 \le n \le \text{NMAX}\}.$$

The quantity p_n is a strictly nonincreasing function of n whereas σ_n typically decreases initially with increasing n, but then oscillates and slowly increases as n becomes so large that the polynomial $p_n(x)$ is fitting noise rather than a true polynomial signal in the data.

The subroutine sets NDEG to be the smallest value of n for which $\sigma^2_n \le 1.01*\text{CMIN}$, and it sets SIGFAC = σ^{NDEG}. This method of selecting NDEG is generally satisfactory when the ratio

NMAX/M is reasonably small, say less than 1/3. If this ratio is larger, there is a tendency for the NDEG selected by this method to be larger than one might choose if one viewed graphs of solution polynomials of different degrees.

Whatever the setting of SEEKN, if the problem for degree NMAX is rank-deficient or very ill-conditioned, the subroutine will restrict consideration to lower degrees for which the problem is not extremely ill-conditioned. In particular this reduction of degree will certainly occur if NMAX+1 > M.

Parameterization of the Polynomial, $p_n(x)$

In this subroutine an n^{th} degree polynomial $p_n(x)$ is represented in one of two special parametric forms (monomial basis or Chebyshev basis), both of which include the specification of a linear transformation of the independent variable in the parameterization. The parameters defining an n^{th} degree polynomial $p_n(x)$ are stored in the array P(I), I=1, ..., n+3.

Using the monomial basis these parameters define the polynomial $p_n(x)$ as follows:

(1) $$u = \frac{x - P(1)}{P(2)}$$

(2) $$P_n(x) = \sum_{i=0}^{n} P(i+3)u^i$$

whereas, if the Chebyshev basis is requested, $p_n(x)$ is defined as:

(3) $$u = \frac{x - P(1)}{P(2)}$$

where $T_i(u)$ denotes the Chebyshev polynomial of degree i. The Chebyshev polynomials may be defined as follows:

(4) $$P_n(x) = \sum_{i=0}^{n} P(i+3)T_i(u)$$

$$T_0(u) = 1, \quad T_1(u) = u, \quad T_2(u) = 2u^2 - 1$$

$$T_i(u) = 2u\, T_{i-1}(u) - T_{i-2}(u) \qquad i=3,4, \ldots$$

The parameters P(1) and P(2) that occur in the transformation formula (1) or (3) may be set by the user (COMTRN = .FALSE.), or else computed by this subroutine (COMTRN = .TRUE.). In the latter case P(1) and P(2) will be computed as

$$P(1) = (x_{max} + x_{min})/2$$

$$P(2) = (x_{max} - x_{min})/2$$

where x_{max} and x_{min} are respectively the maximum and minimum values of x in the data array X(I), I=1, ..., M. This causes the transformed variable u of Eq. (1) or (3) to range from -1 to +1 as x ranges from x_{min} to x_{max}.

Reference

1. C. Lawson and R. Hanson, **Solving Least Squares Problems**, Prentice-Hall (1974).

E. Error Procedures and Restrictions

The use of the Chebyshev basis is effective in improving the conditioning of the problem only if a transformation of the variable is used that maps the interval of interest for both fitting and evaluation to an interval approximately coincident with [-1.,1.].

The automatic procedure for selecting the degree of the fitted polynomial, used when SEEKN = .TRUE., tends to select the degree somewhat higher in some cases than one would choose by looking at plots of fits of different degrees.

This subroutine will issue an error message, set NDEG = -1, and return if any of the following erroneous conditions exists:

1. SD(1) = 0.
2. SD(1) > 0. and some SD(I) < 0. for 1 < I ≤ M.
3. M ≤ 0.
4. NMAX < 0.
5. COMTRN = .FALSE. and P(2) = 0.

F. Supporting Information

The source language is Fortran 77.

Program Unit and Entry Name	**External References**
SPFIT	SROTMG, SROTM, IERM1, IERV1, SERM1, SERV1, R1MACH
DPFIT	DROTMG, DROTM, IERM1, IERV1, DERM1, DERV1, D1MACH

DRSPFIT

```
c     Demonstration driver for SPFIT.
      REAL  X(12), Y(12), P(11), SIGFAC, W(121), SCPVAL
      REAL  SD(1)
      DATA X / 2., 4., 6., 8.,10.,12.,14.,16.,18.,20.,22.,24./
      DATA Y /2.2,4.0,5.0,4.6,2.8,2.7,3.8,5.1,6.1,6.3,5.0,2.0/
      DATA SD(1) / -1.E0 /
      DATA M      / 12 /
      CALL SPFIT(M,X,Y,SD,8, .TRUE. , .TRUE. , .TRUE. ,P,NDEG,SIGFAC,W)
      NP3=NDEG+3
      WRITE(*,1000) NDEG,SIGFAC,(P(I),I=1,NP3)
      WRITE(*,1001)
      DO I=1,M                    ! Using Fortran 90 "DO" syntax.
        YFIT=SCPVAL(P,NDEG,X(I))
        R=Y(I)-YFIT
        WRITE(*,1002)I,X(I),Y(I),YFIT,R
      ENDDO
      STOP
C
 1000 FORMAT('0NDEG =',I2,10X,'SIGFAC =',F8.4//
     *' P(1),P(2) =',9X,2F15.5//' P(3),...,P(NDEG+3) =',3F15.5/
     *(21X,3F15.6))
 1001 FORMAT('0 I    X       Y       YFIT  R=Y-YFIT'/1X)
 1002 FORMAT(1X,I2,F6.0,2F9.3,F10.3)
      END
```

ODSPFIT

```
NDEG = 7          SIGFAC =  0.2216

P(1),P(2) =                13.00000       11.00000

P(3),...,P(NDEG+3) =        3.99473        0.57358       -0.82918
                           -0.583537      -1.423904       0.202192
                            0.356891      -0.298388
```

I	X	Y	YFIT	R=Y-YFIT
1	2.	2.200	2.205	-0.005
2	4.	4.000	3.959	0.041
3	6.	5.000	5.147	-0.147
4	8.	4.600	4.333	0.267
5	10.	2.800	3.028	-0.228
6	12.	2.700	2.699	0.001
7	14.	3.800	3.651	0.149
8	16.	5.100	5.156	-0.056
9	18.	6.100	6.196	-0.096
10	20.	6.300	6.187	0.113
11	22.	5.000	5.048	-0.048
12	24.	2.000	1.992	0.008

11.2 Evaluation, Integration, and Differentiation of Polynomials

A. Purpose

This set of subroutines will evaluate, integrate, or differentiate polynomials. The polynomials may be represented by coefficients relative to either the monomial or Chebyshev basis. The data structure and parameterization used to represent a polynomial is the same as that used by the least squares polynomial curve fit subroutines described in Chapter 11.1.

B. Usage

B.1 Usage for Evaluation

Program Prototype, Single Precision

```
INTEGERNDEG
REAL        P(≥ NDEG+3), X, Y, SCPVAL, SMPVAL
```

Assign values to NDEG, X, and P(I), I = 1, NDEG+3. If the Chebyshev basis is being used, use the statement:

```
Y = SCPVAL (P, NDEG, X)
```

If the monomial basis is being used, use the statement:

```
Y = SMPVAL (P, NDEG, X)
```

Following the appropriate one of these two statements the value of the polynomial at the argument X will be stored in Y.

Argument Definitions

P( ) [in] An array containing NDEG+3 parameters that define a polynomial as described in Section D.

NDEG [in] Degree of the polynomial.

X [in] Argument value at which the polynomial is to be evaluated.

SCPVAL [out] The value of the polynomial evaluated at X assuming the Chebyshev basis representation.

SMPVAL [out] The value of the polynomial evaluated at X assuming the monomial basis representation.

B.2 Usage for Integration

Program Prototype, Single Precision

```
INTEGERNDEGA, NDEGB
REAL        A(≥ NDEGA+3), B(≥ NDEGA+4)
```

Assign values to NDEGA and A(I), I = 1, ..., NDEGA+3. If the Chebyshev basis is being used, use the statement:

```
CALL SCPINT (A, NDEGA, B, NDEGB)
```

If the monomial basis is being used, use the statement:

```
CALL SMPINT (A, NDEGA, B, NDEGB)
```

Following the appropriate one of these two call statements the results will be stored in B( ) and NDEGB.

Argument Definitions

A( ) [in] An array containing NDEGA+3 parameters that define the input polynomial, say p(x). See Section D for the specification of the parameterization.

NDEGA [in] Degree of the input polynomial, p(x).

B( ) [out] On return, B( ) will contain NDEGB+3 parameters defining the output polynomial, say q(x), which is the indefinite integral of the input polynomial p(x). Mathematically the constant term of q(x) is an arbitrary constant of integration. This subroutine will set the constant term, B(3), to zero. The storage locations occupied by A( ) and B( ) must be distinct.

NDEGB [out] The subroutine sets NDEGB = NDEGA+1 to indicate the degree of the output polynomial. The storage locations occupied by NDEGA and NDEGB must be distinct.

B.3 Usage for Differentiation

Program Prototype, Single Precision

```
INTEGERNDEGC, NDEGD
REAL          C(≥ NDEGC+3), D(≥ max [3, NDEGC+2])
```

Assign values to NDEGC and C(I), I = 1, ..., NDEGC+3. If the Chebyshev basis is being used, use the statement:

```
CALL SCPDRV (C, NDEGC, D, NDEGD)
```

If the monomial basis is being used, use the statement:

```
CALL SMPDRV (C, NDEGC, D, NDEGD)
```

Following the appropriate one of these two call statements the results will be stored in D( ) and NDEGD.

Argument Definitions

C( ) [in] An array containing NDEGC+3 parameters that define the input polynomial, say p(x). See Section D for the specification of the parameterization.

NDEGC [in] Degree of the input polynomial, p(x).

D( ) [out] On return, D( ) will contain NDEGD+3 parameters defining the output polynomial, say q(x), which is the derivative of the input polynomial p(x). The storage locations occupied by C( ) and D( ) must be distinct.

NDEGD [out] The subroutine sets NDEGD = max [0, NDEGC-1] to indicate the degree of the output polynomial. The storage locations occupied by NDEGC and NDEGD must be distinct.

B.4 Usage for Double Precision Evaluation, Integration or Differentiation

For DOUBLE PRECISION usage, change the REAL type statements to DOUBLE PRECISION and change the initial "S" of the function and subroutine names to a "D." Note particularly that if the function names DCPVAL or DMPVAL are used, they must be typed DOUBLE PRECISION either explicitly or via an IMPLICIT statement.

C. Examples and Remarks

Let a cubic polynomial p(x) be defined relative to the Chebyshev basis as $p(x) = 10 + 8T_1(u) + 6T_2(u) + 4T_3(u)$ where $u = (x - 5)/2$. The DRSCPVAL program computes the indefinite integral of p(x) calling it q(x). This computation is checked by computing r(x) as the derivative of q(x). Note that r(x) agrees with p(x). Finally the program evaluates the definite integral:

$$z = \int_4^6 p(x)dx = q(6) - q(4) = 10.$$

The output from this program is shown in ODSCPVAL.

D. Functional Description

In typical expected usage the polynomial parameter vector input to any of the subprograms of this set will have been produced by the library curve fitting subroutine SPFIT (or DPFIT) or an integration or differentiation subroutines of this set. The subprograms of this set are thus intended to let the user do the operations of evaluation, integration or differentiation of polynomials without being concerned with the details of the parametric representation of polynomials or algorithmic details.

The following description is provided for those who wish to know more of the internal details.

For the purposes of this set of subprograms, a polynomial of degree n, say p(x), is represented by a set of n + 3 parameters, say a_1, ..., a_{n+3}. The first two parameters define a linear transformation of the independent variable:

$$u = (x - a_1)/a_2.$$

The remaining n + 1 parameters are coefficients of an n^{th} degree polynomial in the transformed variable u. If the Chebyshev basis is used this polynomial is

$$p(x) = q(u) = \sum_{i=0}^{n} a_{i+3} T_i(u)$$

whereas if the monomial basis is used the polynomial is

$$p(x) = q(u) = \sum_{i=0}^{n} a_{i+3} u^i.$$

The Chebyshev polynomials $T_i(u)$ are defined by the equations:

$$T_0(u) = 1$$

$$T_1(u) = u$$

$$T_i(u) = 2uT_{i-1}(u) - T_{i-2}(u) \qquad i = 2, 3, \ldots$$

The formulas for differentiation and integration of polynomials expressed using the Chebyshev basis may be derived from the following standard identities:

$$0 = dT_0(u)/du$$

$$T_0(u) = dT_1(u)/du$$

$$T_1(u) = 0.25\, dT_2(u)/du$$

$$T_i(u) = \frac{1}{2}\frac{d}{du}\left[\frac{T_{i+1}(u)}{i+1} - \frac{T_{i-1}(u)}{i-1}\right],\ i = 2,3,...$$

The algorithms used by this set of subprograms are specified as follows:

Monomial Basis Evaluation, SMPVAL or DMPVAL

Given an n^{th} degree polynomial p represented by the parameters p_i, i = 1, ..., n + 3 and an argument x, compute y = p(x).

$$u = (x-p_1)/p_2$$

$$z_n = p_{n+3}$$

$$z_i = uz_{i+1} + p_{i+3} \qquad i = n-1, n-2, ..., 0$$

$$y = z_0$$

Monomial Basis Integration, SMPINT or DMPINT

Given an n^{th} degree polynomial p represented by the parameters a_i, i = 1, ..., n + 3, compute the parameters b_i that represent a polynomial q that for arbitrary u and v satisfies:

$$\int_u^v p(x)dx = q(v) - q(u).$$

The formulas used are:

$$b_1 = a_1,\ b_2 = a_2,\ b_3 = 0$$

$$b_{i+3} = a_2 a_{i+2}/i \qquad i = 1, ..., n + 1.$$

Monomial Basis Differentiation, SMPDRV or DMPDRV

Given an n^{th} degree polynomial p represented by the parameters c_i, i = 1, ..., n + 3, compute the parameters d_i that represent the polynomial q satisfying:

$$(d/dx)\, p(x) = q(x).$$

The formulas used are:

$$d_1 = c_1,\ d_2 = c_2$$

$$d_{i+3} = (i + 1)c_{i+4}/c_2 \qquad i = 0, ..., n - 1$$

with a special case of $d_3 = 0$ if n = 0.

Chebyshev Basis Evaluation, SCPVAL or DCPVAL

Given an n^{th} degree polynomial p represented by the parameters p_i, i = 1, ..., n + 3 and an argument x, compute y = p(x).

$$u = (x - p_1)/p_2$$

$$z_n = p_{n+3}$$

$$z_{n-1} = 2uz_n + p_{n+2}$$

$$z_i = 2uz_{i+1} - z_{i+2} + p_{i+3} \quad i = n - 2, ..., 1$$

$$y = uz_1 - z_2 + p_3$$

Chebyshev Basis Integration, SCPINT or DCPINT

Given an n^{th} degree polynomial p represented by the parameters a_i, i = 1,..., n + 3, compute the parameters b_i that represent a polynomial q that for arbitrary u and v satisfies:

$$\int_u^v p(x)dx = q(v) - q(u).$$

The formulas used are:

$$b_1 = a_1,\ b_2 = a_2,\ b_3 = 0$$

$$b_4 = a_2[a_3 - (1/2)a_5]$$

$$b_{i+3} = a_2(a_{i+2} - a_{i+4})/(2i) \qquad i = 2, \ldots, n + 1$$

where a_i for i > n + 3 is taken to be zero.

Chebyshev Basis Differentiation, SCPDRV or DCPDRV

Given an n^{th} degree polynomial p represented by the parameters c_i, i = 1, ..., n + 3, compute the parameters d_i that represent the polynomial q satisfying:

$$(d/dx)\,p(x) = q(x).$$

The formulas used are:

$$d_1 = c_1,\ d_2 = c_2$$

$$d_{i+3} = 2(i + 1)c_{i+4}/c_2 \qquad i = n - 1, n - 2$$

$$d_{i+3} = d_{i+5} + 2(i + 1)c_{i+4}/c_2 \qquad i = n - 3, \ldots, 0$$

$$d_3 = d_3/2$$

with a special case of $d_3 = 0$ if n = 0.

E. Error Procedures and Restrictions

The degree of the input polynomial must be zero or positive. If it is negative, the subprograms in this set will issue an error message and return taking no action.

The given values of P(2), A(2), and C(2) must be nonzero.

The storage locations for the input quantities in the integration and differentiation subroutines must be distinct from the storage locations for the output quantities.

Since DCPVAL and DMPVAL are FUNCTION type subprograms, their names must be typed DOUBLE PRECISION in any program that uses them.

F. Supporting Information

The source language is Fortran 77. The program units and entry names are

SCPVAL, SCPINT, SCPDRV, SMPVAL, SMPINT, SMPDRV,
DCPVAL, DCPINT, DCPDRV, DMPVAL, DMPINT, DMPDRV

The 'INT' and 'DRV' program units each have one external reference to the error message processor, IERM1.

DRSCPVAL

```
C     DEMONSTRATE SCPDRV, SCPINT, AND SCPVAL.
C
      REAL     P(6),Q(7),R(6)
      DATA     P/ 5.E0, 2.E0, 10.E0, 8.E0, 6.E0, 4.E0 /
      CALL SCPINT(P,3,Q,NQ)
      CALL SCPDRV(Q,NQ,R,NR)
      Z=SCPVAL(Q,NQ,6.E0)-SCPVAL(Q,NQ,4.E0)
      WRITE(*,1000) P,Q,R,Z
      STOP
 1000 FORMAT(21X,'P =',2F4.0,2X,4F7.2//' INTEGRAL OF P.      Q ='
     *,2F4.0,2X,5F7.2//' DERIVATIVE OF Q.    R =',2F4.0,2X,4F7.2/
     */' DEFINITE INTEGRAL.  Z =',F20.5)
            END
```

ODSCPVAL

```
                  P =  5.  2.    10.00   8.00   6.00   4.00
INTEGRAL OF P.    Q =  5.  2.     0.00  14.00   2.00   2.00   1.00
DERIVATIVE OF Q.  R =  5.  2.    10.00   8.00   6.00   4.00
DEFINITE INTEGRAL.  Z =           10.00000
```

11.3 Conversion between Chebyshev and Monomial Representations of a Polynomial

A. Purpose

These subroutines convert a polynomial represented in the monomial basis to a representation in the Chebyshev basis, and vice versa.

B. Usage

B.1 Program Prototype, Single Precision

```
INTEGERN
REAL        COEFF(0:≥N)
```

Assign values to N, and to coefficients in COEFF. If COEFF(i) contains coefficients of $T_i(x)$, i = 0, 1, ..., N, which are to be converted to coefficients of x^i,

```
CALL SCONCM(N, COEFF)
```

For the inverse operation,

```
CALL SCONMC(N, COEFF)
```

Argument Definitions

N [in] The degree of the polynomial.

COEFF [inout] When calling SCONCM, COEFF(i) contains the coefficient of T_i, i = 0, 1, ... N, on input, and contains the coefficient of x^i on output. When calling SCONMC, COEFF(i) contains the coefficient of x^i, i = 0, 1, ..., N on input, and the coefficient of T_i on output.

Modifications for Double Precision

Change the names SCONCM and SCONMC to DCONCM and DCONMC respectively, and change the REAL declaration to DOUBLE PRECISION.

C. Examples and Remarks

The program DRSCON prints out the coefficients of the Chebyshev polynomials corresponding to x^k, k = 0, 1, ..., 6, and then prints the coefficients in the monomial basis corresponding to the Chebyshev polynomials T_k, k = 0, 1, ..., 6. Results are in the file ODSCON.

If these subroutines are applied to a coefficient array, say P(), obtained from SPFIT, Chapter 11.1, the zeroth order coefficient is in P(3), so the call would be of the form SCONxx(NDEG, P(3)), where xx is either CM or MC.

D. Functional Description

Consider the polynomial $p_n(x)$ of degree n, which can be written as:

$$(1) \qquad p_n(x) = \sum_{k=0}^{n} a_k x^k \;/\; \sum_{k=0}^{n} c_k T_k(x)$$

where $T_k(x)$ is the kth Chebyshev polynomial $\equiv \cos(k \cos^{-1} x)$. This software converts between the a_k's and the c_k's.

The method used is based on the well-known identities for the Chebyshev polynomials,

$$xT_k(x) = \frac{1}{2}\left[T_{k+1}(x) + T_{k-1}(x)\right]$$

(2)

$$xT_0(x) = T_1(x) = x.$$

We can write p_n in forms intermediate between the extremes represented in Eq. (1). It is these intermediate forms that are used in obtaining the recurrences. Thus,

$$p_n(x) = \sum_{k=0}^{j-1} a_k x^k + x^j \sum_{k=0}^{n-j} b_{k,j} T_k(x) \;/\; \sum_{k=0}^{j} a_k x^k + x^{j+1} \sum_{k=0}^{n-j-1} b_{k,j+1} T_k(x). \tag{3}$$

Note that $b_{k,0} \equiv c_k$. Using Eq. (2), Eq. (3) can be replaced by:

$$p_n(x) = \sum_{k=0}^{j-1} a_k x^k + \frac{x^j}{2} \sum_{k=1}^{n-j-1} b_{k,j+1}\left[T_{k+1}(x) + T_{k-1}(x)\right] + x^j b_{0,j+1} T_1(x). \tag{4}$$

Collecting like terms in Eqs. (3) and (4), we obtain,

$$\begin{aligned}
a_j + \frac{1}{2} b_{1,j+1} &= b_{0,j} \\
b_{0,j+1} + \frac{1}{2} b_{2,j+1} &= b_{1,j} \\
\frac{1}{2}\left[b_{k1,j+1} + b_{k+1,j+1}\right] &= b_{k,j}, \quad k = 2,3,\ldots,n-j-1 \\
\frac{1}{2} b_{n-j-1,j+1} &= b_{n-j,j}.
\end{aligned} \tag{5}$$

The computation becomes more efficient if we replace $b_{k,j}$ with $2^j B_{k,j}$. With this replacement, we obtain the result,

$$\begin{aligned}
2^{-j} a_j + B_{1,j+1} &= B_{0,j} \\
B_{0,j+1} + B_{2,j+1} &= B_{1,j} \\
B_{k-1,j+1} + B_{k+1,j+1} &= B_{k,j}, \quad k = 2,3,n-j-1 \\
B_{n-j-1,j+1} &= B_{n-j,j}.
\end{aligned} \tag{6}$$

In the code, the $B_{j-k,k}$ share space with the original a_k or the original c_k. If one starts with the a_k then one runs j from n down to 0, otherwise, j runs in the opposite direction. Observe that the innermost loop requires only a single addition.

E. Error Procedures and Restrictions

If n < 0, a return is made without taking any action.

F. Supporting Information

The source language is ANSI Fortran 77.

Program Unit and Entry Names: SCONCM, SCONMC, DCONCM, DCONMC

There are no external references.

DRSCON

```
      program DRSCON
c     Check program for converting between Chebyshev and monomial basis.
c
      parameter (NMAX=6)
      integer          N
      real             COEFF(0:NMAX)
c
      print '(7X, 9(:''    X**'',I1))', (K, K = 0, NMAX)
      do  N = 0, NMAX
         do  K = 0, N-1
            COEFF(K) = 0.E0
         end do
         COEFF(N) = 1.E0
         call SCONCM(N, COEFF)
         print '('' T'', I1, ''(X) ='', F7.3, 8F8.3)', N,
     1      (COEFF(K), K = 0, N)
      end do
      print '(''0'', 5X, 9(:''   T'', I1, ''(X)''))', (K, K = 0, NMAX)
      do  N = 0, NMAX
         do  K = 0, N-1
            COEFF(K) = 0.E0
         end do
         COEFF(N) = 1.E0
         call SCONMC(N, COEFF)
         print '('' X**'', I1, '' ='', 9F8.5)', N, (COEFF(K), K = 0, N)
      end do
      stop
      end
```

ODSCON

```
          X**0    X**1    X**2    X**3    X**4    X**5    X**6
T0(X) =  1.000
T1(X) =  0.000   1.000
T2(X) = -1.000   0.000   2.000
T3(X) =  0.000  -3.000   0.000   4.000
T4(X) =  1.000   0.000  -8.000   0.000   8.000
T5(X) =  0.000   5.000   0.000 -20.000   0.000  16.000
T6(X) = -1.000   0.000  18.000   0.000 -48.000   0.000  32.000

        T0(X)   T1(X)   T2(X)   T3(X)   T4(X)   T5(X)   T6(X)
X**0 = 1.00000
X**1 = 0.00000 1.00000
X**2 = 0.50000 0.00000 0.50000
X**3 = 0.00000 0.75000 0.00000 0.25000
X**4 = 0.37500 0.00000 0.50000 0.00000 0.12500
X**5 = 0.00000 0.62500 0.00000 0.31250 0.00000 0.06250
X**6 = 0.31250 0.00000 0.46875 0.00000 0.18750 0.00000 0.03125
```

11.4 Least-Squares Cubic Spline Fit

A. Purpose

A cubic spline function with NB-1 segments is a function consisting of NB-1 pieces, each of which is a cubic polynomial. At the abscissae, called knots, at which adjacent segments meet, the function has C^2 continuity, i.e. continuity in value, first derivative, and second derivative.

Subroutine SC2FIT or DC2FIT will determine the (NB-1)-segment cubic spline function, with user specified knots, that best fits a set of discrete data in the sense of weighted least squares, and return the values of the fitted spline curve and its first derivative at the knots. The user can then evaluate the curve, or its first or second derivative, at any argument using the Hermite interpolation subroutine, SHINT or DHINT, of Chapter 12.

This software can be used for interpolation by setting the number of knots, NB, to be two less than the number of data points, NXY. Setting NB < NXY - 2 gives least-squares approximation.

B. Usage

Program Prototype, Single Precision

```
INTEGERNXY, NB, LDW, IERR1
REAL        X(≥ NXY), Y(≥ NXY), SD(≥ NXY), B(≥ NB)
            W(LDW,5), YKNOT(≥ NB), YPKNOT(≥ NB), SIGFAC
```

Assign values to X(), Y(), SD(), NXY, B(), NB, and LDW.

```
CALL SC2FIT(X, Y, SD, NXY, B, NB, W, LDW, YKNOT, YPKNOT, SIGFAC, IERR1)
```

Computed quantities are returned in YKNOT(), YPKNOT( ), SIGFAC, and IERR1. Following the use of SC2FIT, the user may use SHINT of Chapter 12 to compute values of the fitted curve.

Argument Definitions

X(),Y() [in] Data pairs (X(I), Y(I), I=1, ..., NXY). The contents of X() must satisfy $X(1) \le X(2) \le \ldots \le X(NXY)$.

SD() [in] If SD(1) > 0., each SD(I) must be positive and must be the user's a priori estimate of the standard deviation of the uncertainty (e.g., observational error) in the corresponding data value Y(I).

If SD(1) < 0., |SD(1)| will be used as the a priori standard deviation of each data value Y(I). In this case the array SD() may be dimensioned as SD(1).

An error condition is reported if SD(1) = 0.

NXY [in] Number of data points. Require NXY ≥ 4.

B() [in] Set by the user to specify the knot abscissae and endpoints for the spline curve. Must satisfy B(1) < B(2) < ... < B(NB). Also require $B(1) \le \min_i [X(i)]$ and $B(NB) \ge \max_i [X(i)]$.

NB [in] Number of knots, including endpoints. The number of segments in the spline curve will be NB-1. The number of degrees of freedom for the fit will be NB+2. Require $2 \le NB \le NXY-2$.

W(,) [scratch] Working space, dimensioned W(LDW,5).

LDW [in] Leading dimension for the work array W(). LDW must be at least NB + 4, but the execution time is less for larger values of LDW, up to NB + 3 + k, where k is the largest number of data points lying between any adjacent pair of knots.

YKNOT(), YPKNOT() [out] Arrays, each of length at least NB, in which the subroutine will store a definition of the fitted spline curve as a sequence of values of the curve and its first derivative at the knot abscissae B(I). Letting f denote the fitted curve, the elements of these arrays will be set to:

YNOT(i) = f(B(i)), i=1, ..., NB
YPNOT(i) = f'(B(i)), i=1, ..., NB.

SIGFAC [out] Set by the subroutine as a measure of the residual error of the fit. See discussion in Section D.

IERR1 [out] Error status indicator. Set on the basis of tests done in SC2FIT as well as error indicators IERR2 set by SBACC and IERR3 set by SBSOL. Zero indicates no errors detected. See Section E for the meaning of nonzero values.

Modifications for Double Precision

For double precision usage change the REAL statement to DOUBLE PRECISION and change the subroutine name SC2FIT to DC2FIT.

C. Examples and Remarks

Example: Given a set of 12 data pairs (x_i,y_i), compute the uniformly weighted least squares cubic spline fit to these data using six uniformly spaced breakpoints, including endpoints. After determining the spline function f(x), compute and tabulate the quantities x_i, y_i, $f(x_i)$, and $r_i = y_i - f(x_i)$.

This computation is illustrated by the program DRDC2FIT and the output ODDC2FIT. The fitted curve is determined using DC2FIT, and is evaluated using DHINT of Chapter 12.

Interpolation: If all of the data abcissae are distinct, and one wants interpolation, rather than the smoothing effect of least-squares approximation, one can choose interpolation by setting NB = NXY - 2. The NB knot abcissae can be assigned in various ways, but one reasonable way is to set B(1) = X(1), B(i) = X(i+1), for i = 2, ..., NB-1, and B(NB) = X(NXY).

D. Functional Description

Let knot abscissae $b_1 < b_2 < ... < b_{NB}$ be given. A cubic spline function defined over the interval $[b_1, b_{NB}]$ is a cubic polynomial in each subinterval $[b_i, b_{i+1}]$, i = 1, ..., NB-1, with continuity of the value, first derivative, and second derivative at each internal knot, b_2, ..., b_{NB-1}. The set of all cubic spline functions defined relative to this knot set is a linear space of dimension d = NB+2. If the knot spacing does not depart severely from uniformity, a well conditioned set of basis functions for this space is provided by a particular set of cubic spline functions called B-splines, $p_i(x)$, i = 1, ..., d, each of which is nonzero over at most four adjacent subintervals [Ref 1].

The problem data is $\{(x_i, y_i, s_i), i = 1, ..., NXY\}$, where s_i is the a priori standard deviation of the error in the value y_i. The weighted least-squares curve fitting problem then becomes one of determining coefficients c_j to minimize:

$$p^2(c) = \sum_{i=1}^{NXY} \left[\frac{y_i - \sum_{j=1}^{d} c_j p_j(x_i)}{s_i} \right]^2 .$$

The matrix formulation of this least-squares problem involves a matrix having a banded form in which at most four elements are nonzero in each row. This least-squares problem is solved using the subroutines of Chapter 4.5

This problem will have a unique set of solution coefficients, c_j, if NB ≤ NXY-2 and the positioning of the knots is such that there exists an indexing of some set of NB+2 of the distinct data abscissae, x_i (not necessarily the indexing used in the subprogram) such that $b_{i-3} < x_i < b_{i+1}$ for i = 1, ..., NB+2. Here b_{-2}, b_{-1}, and b_0 denote fictitious knots to the left of b_1, and b_{NB+1}, b_{NB+2}, and b_{NB+3} denote fictitious knots to the right of b_{NB} [Ref 2]. If the solution is not unique, no solution will be given and an error code will be returned as described in Section E.

After determining coefficients, c_j, SC2FIT uses subroutine STRC2C to evaluate the value and first derivative of the fitted curve at the knots. These quantities are returned to the user in the arrays YKNOT() and YPKNOT() as the defining parameters of the fitted curve.

Subroutine SC2BAS is called by both SC2FIT and STRC2C to evaluate B-spline basis functions.

References

1. Carl de Boor, *On Calculating with B-Splines*, **Jour. Approximation Theory 6**, 1 (1972) pp 50-62. Also: C. de Boor, **A Practical Guide to Splines**, Springer-Verlag (1978) 392 pp.
2. John Rice, *Characterization of Chebyshev Approximations by Splines*, **SIAM J. Numer. Anal. 4**, 4 (1967) pp 557-565.

E. Error Procedures and Restrictions

SC2FIT sets IERR1 and issues error messages based on internal tests as well as propagating error information set in IERR2 by SBACC and in IERR3 by SBSOL. See Chap. 4.5 for the meaning of IERR2 and IERR3. In all cases in which IERR1 is set nonzero, no solution will be computed.

IERR1	Meaning
0	No errors detected.
100	NB < 2 or NXY < NB+2
200	B(I) ≥ B(I+1) for some I.
300	LDW < NB+4
400	X(I-1) > X(I) for some I.
500	B(1) > X(1) or B(NB) < X(NXY)
600	Need larger dimension LDW.
700 + IERR2	Set when SBACC returns with IERR2 ≠ 0
800 + IERR2	Set when SBACC returns with IERR2 ≠ 0
900 + IERR2	Set when SBACC returns with IERR2 ≠ 0
1000 + IERR3	Set when SBSOL returns with IERR3 ≠ 0. Indicates singularity.
1100	SD(1) = zero.

F. Supporting Information

Program Unit and Entry Name	Source Language	External References

DC2FIT	SFTRAN3DBACC, DBSOL, DC2BAS, DTRC2C, ERMSG, IERM1, IERV1, DERM1, DERV1	
DTRC2C	Fortran 77	DC2BAS
DC2BAS	Fortran 77	None
SC2FIT	SFTRAN3SBACC, SBSOL, SC2BAS, STRC2C, ERMSG, IERM1, IERV1, SERM1, SERV1	
STRC2C	Fortran 77	SC2BAS
SC2BAS	Fortran 77	None

DRDC2FIT

```
c     program DRDC2FIT
c     Demonstration driver for DC2FIT.
c     ----------------------------------------------------------------
      parameter(NXY = 12, NB = 6, NW = 10)
      double precision DHINT, X(NXY), Y(NXY), SD(01), B(NB), W(NW,5)
      double precision YKNOT(NB), YPKNOT(NB), R, SIGFAC, YFIT
      integer IERR
      data X / 2.D0, 4.D0, 6.D0, 8.D0,10.D0,12.D0,
     *        14.D0,16.D0,18.D0,20.D0,22.D0,24.D0/
      data Y /2.2D0,4.0D0,5.0D0,4.6D0,2.8D0,2.7D0,
     *        3.8D0,5.1D0,6.1D0,6.3D0,5.0D0,2.0D0/
      data B / 2.0D0, 6.4D0, 10.8D0, 15.2D0, 19.6D0, 24.0D0 /
      data NDERIV / 0 /    SD(1) / -1.0D0 /
c     ----------------------------------------------------------------
      call DC2FIT(X, Y, SD, NXY, B, NB, W, NW, YKNOT,YPKNOT,SIGFAC,IERR)
      write(*,1000)  IERR, SIGFAC, (YKNOT(I),I=1,NB)
      write(*,1001)  (YPKNOT(I),I=1,NB)
      write(*,1002)
      do I=1,NXY            !  Using Fortran 90 "DO" syntax.
         YFIT=  DHINT(X(I), NDERIV, NB, B, YKNOT, YPKNOT)
         R=Y(I)-YFIT
         write(*,1003)I,X(I),Y(I),YFIT,R
      enddo
      stop
 1000 format(1X/  5X,'IERR =',I5, ',  SIGFAC =', F10.5
     * //    1x,' YKNOT() = ',4F10.5/(12X,4F10.5))
 1001 format(1x,'YPKNOT() = ',4f10.5/(12x,4f10.5))
 1002 format( '0 I    X       Y       YFIT  R=Y-YFIT'/1X)
 1003 format(1X,I2,F6.0,2F9.3,F10.3)
  end
```

ODDC2FIT

```
      IERR =    0,       SIGFAC =   0.14664
   YKNOT() =    2.20672   5.13370   2.61122   4.61735
                6.30079   1.99475
  YPKNOT() =    0.76829  -0.05990  -0.25290   0.71946
               -0.10933  -2.07029
```

I	X	Y	YFIT	R=Y-YFIT
1	2.	2.200	2.207	-0.007
2	4.	4.000	3.958	0.042
3	6.	5.000	5.111	-0.111
4	8.	4.600	4.430	0.170
5	10.	2.800	2.959	-0.159
6	12.	2.700	2.646	0.054
7	14.	3.800	3.734	0.066
8	16.	5.100	5.162	-0.062
9	18.	6.100	6.132	-0.032
10	20.	6.300	6.233	0.067
11	22.	5.000	5.033	-0.033
12	24.	2.000	1.995	0.005

11.5 Least-Squares Data Fitting using Kth Order Splines with Constraints

A. Purpose

This package contains two subprograms, DSFIT and DSFITC, for fitting a *polynomial spline function* to discrete data. A *polynomial spline function* is a piecewise polynomial function having specified orders of continuity at the abcissae, called *knots*, at which one polynomial piece ends and another begins. Spline functions have been found to be very useful in many computational processes due to their capability of representing a wide variety of shapes in a controlled way.

DSFIT can be used either for a weighted least squares fit or for interpolation. DSFITC adds capabilities for the user to specify constraints on the fit in the form of equality or inequality conditions on the value or derivative (of specified order), of the spline function at specified points, or the integral of the spline function over a specified interval. These constraints can be used, for example, to assure monotonicity or convexity of the fitted spline function over specified intervals.

The fitting subprograms return coefficients of a spline function relative to *B-spline basis functions* using the parameterization conventions given by Carl de Boor in Ref 1. In this approach the spline function will be of a user-specified order, K, which means the polynomial pieces are of degree at most K-1. By default the continuity at *knots* will be of order K-2, however the user can specify lower order of continuity at selected knots to allow the curve to change direction more sharply.

For a spline function represented relative to the B-spline basis, subprogram DSVAL can be used to evaluate the function or any of its derivatives at a specified point, and subprogram DSQUAD can be used to evaluate the definite integral of the function between specified limits.

The B-spline representation has the desirable property that usual continuity conditions at knots are "built-in". This keeps down the number of coefficients that must be determined in interpolation or least-squares fitting. A disadvantage is that evaluation of a function represented in the B-spline representation is more expensive than is the use of an alternative representation using the *power* basis.

If one is going to do a large number of evaluations of a spline function, one may choose to convert the representation of the spline function from the B-spline basis to the power basis to allow for more efficient evaluation. Subprogram DSTOP can be used to do this conversion. Then DPVAL can be used to evaluate the function or any of its derivatives at a specified point, and DPQUAD can be used to evaluate the definite integral of the function between specified limits.

B. Usage

B1. Usage of DSFIT for Fitting Without Constraints

Program Prototype, Double Precision

```
INTEGER            NXY, KORDER, NCOEF, LDW, IERR1
DOUBLE PRECISION   BCOEF(≥NCOEF), X(≥NXY), Y(≥NXY), SD(≥NXY),
                   TKNOTS (≥NCOEF+KORDER), W(LDW,5), SIGFAC
```

Assign values to X(), Y(), SD(), NXY, KORDER, NCOEF, TKNOTS(), and LDW.

```
CALL DSFIT(X, Y, SD, NXY, KORDER, NCOEF, TKNOTS, BCOEF, SIGFAC, IERR1, LDW, W)
```

Computed quantities are returned in BCOEF(), SIGFAC, and IERR1. Following use of DSFIT, the user may use DSVAL to compute values or specified derivatives of the fitted curve, DSQUAD to compute the definite integral of the fitted curve over a specified interval, or DSTOP to convert the representation to the *power* basis.

Argument Definitions

X(),Y() [in] Data pairs (X(i), Y(i), i=1, ..., NXY). Must be ordered so the X(i)'s are either nondecreasing or nonincreasing.

SD() [in] If SD(1) > 0., each SD(i) must be positive and must be the user's a priori estimate of the standard deviation of the uncertainty (e.g., observational error) in the corresponding data value Y(i).

If SD(1) < 0., |SD(1)| will be used as the a priori standard deviation of each data value Y(I). In this case the array SD() may be dimensioned as SD(1).

An error condition is reported if SD(1) = 0.

NXY [in] Number of data points. Require NXY ≥ max(NCOEF, KORDER).

KORDER [in] Order of the spline function. Each polynomial piece will be of degree at most KORDER-1. The default order of continuity at each internal knot will be KORDER-2. The popular case of a cubic spline with C^2 continuity at the knots is selected by setting KORDER = 4. Require KORDER ≥ 1. Internal arrays in this package impose an upper limit of *kmax* = 20 on KORDER.

NCOEF [in] Number of terms in the sum representing the spline function. Require NCOEF ≤ NXY.

TKNOTS() [in] The knots, t_i, i = 1, ..., NCOEF+KORDER. The interval [t_{KORDER}, $t_{NCOEF+1}$] will be the *proper interpolation interval* for the problem. This interval should contain all the X(j) values, so it is reasonable to set $t_{KORDER} \le \min(X(1), X(NXY))$ and $t_{NCOEF+1} \ge \max(X(1), X(NXY))$. It is convenient and reasonable to set the KORDER-1 knots with indices less than KORDER equal to t_{KORDER}, and the KORDER-1 knots with indices greater than NCOEF+1 equal to $t_{NCOEF+1}$.

Knots indexed from KORDER+1 through NCOEF are *internal* knots. Internal knots specify abcissae at which one polynomial piece ends and the next begins. Successively indexed internal knots may have the same value. A knot appearing with multiplicity μ means the order of continuity of the spline at this knot will be at least KORDER-μ-1. Require 1 ≤ μ ≤ KORDER.

Require $t_i \le t_{i+1}$ for i = 1, ..., NCOEF+KORDER-1; $t_i < t_{i+KORDER}$ for i = 1, ..., NCOEF; $t_{KORDER} < t_{KORDER+1}$; and $t_{NCOEF} < t_{NCOEF+1}$. See Sections C and D for further discussion of knot placement.

BCOEF() [out] Coefficients c_i, i = 1, NCOEF, in the sum representing the spline function as a sum of coefficients times B-spline basis functions.

SIGFAC [out] Set by the subroutine as a measure of the residual error of the fit. The subroutine sets:

$$SIGFAC := \frac{RNORM}{DOF^{1/2}}, \text{ where } RNORM := \left[\sum_{i=1}^{NXY} \left(\frac{yfit_i - Y_i}{SD_i} \right)^2 \right]^{1/2}, \text{ and } DOF := \max(1, NXY - NC).$$

Here SD_i denotes SD(*i*) if SD(1) > 0, and |SD(1)| otherwise.

IERR1 [out] Error status indicator. Set on the basis of tests done in DSFIT, as well as error indicators IERR2 set by DBACC and IERR3 set by DBSOL.

IERR1 = 0 means no errors detected.
= 100 means NC < 1 or NC > NXY
= 150 means KORDER > *kmax* (= 20)
= 200 means TKNOTS(I) > TKNOTS(I+1)
= 250 means TKNOTS(I) ≥ TKNOTS(I+KORDER)
= 300 means LDW < NCOEF + 2
= 400 means The X(*i*)'s are neither nondecreasing nor nonincreasing.
= 600 means LDW < NCOEF + 2.
= 700 + IERR2 means IERR2 ≠ 0
= 800 + IERR2 means IERR2 ≠ 0
= 900 + IERR2 means IERR2 ≠ 0
= 1000 + IERR3 means IERR3 ≠ 0 due to singularity detected in DBSOL.
= 1100 means SDI(1) = zero.

LDW [in] Leading dimension for the work array W(). Require LDW ≥ NCOEF + 2. Let α denote the maximum number of data abcissae, X(*i*), in any one knot interval, i.e. between TKNOTS(*j*) and TKNOTS(*j*+1) for some *j*. The subroutine will be more efficient if LDW is at least NCOEF+1+α.

W(,) [scratch] Working space, dimensioned W(LDW,≤KORDER+1).

B2. Usage of DSFITC for Fitting with Constraints

Program Prototype, Double Precision

```
INTEGER             KORDER, NCOEF, ISET(3), INFO(ninfo)
DOUBLE PRECISION    X(mdim), Y(mdim), SD(mdim),
TKNOTS(≥NCOEF+KORDER),
                    BCOEF(≥NCOEF), RNORM, W(nwork)
CHARACTER*4         CCODE(mdim)
```

The dimension *mdim* must be large enough to provide for specification of all constraint and least-squares equations as described below in the description of CCODE(). See ISET() for the specifications of *ninfo* and *nwork*.

Assign values to CCODE(), X(), Y(), SD(), KORDER, NCOEF, TKNOTS(), and ISET().

```
CALL DSFITC(CCODE, X, Y, SD, KORDER, NCOEF, TKNOTS, BCOEF, RNORM, ISET, INFO, W)
```

Computed quantities are returned in BCOEF(), RNORM, and INFO(). Following use of DSFITC, the user may use DSVAL to compute values or specified derivatives of the fitted curve, DSQUAD to compute the definite integral of the fitted curve over a specified interval, or DSTOP to convert the representation to the *power* basis.

Argument Definitions

CCODE() [in, character*4] CCODE(*i*), or in some cases CCODE(*i*) and CCODE(*i*+1) together, give specifications for one constraint equation or one least-squares equation. CCODE(*i*) is regarded as consisting of four single-character fields.

CCODE(*i*)(1:1) = $kind_i$ = '1', '2', '3', '4'.
CCODE(*i*)(2:2) = $deriv_i$ = '0', '1', ..., '9'.
CCODE(*i*)(3:3) = $relop_i$ = '~', '=', '<', '>'.
CCODE(*i*)(4:4) = $active_i$ = 'A', 'N', '!'.

Where alphabetic characters are shown, the corresponding lower case character is also acceptable.

$active_i$ = '!' signals the end of information in this array. The user must provide this termination signal. The other fields in this array element will be ignored. $active_i$ = 'A' means CCODE(*i*) is active so CCODE(*i*) will be processed. $active_i$ = 'N' means CCODE(*i*) is inactive so processing will advance to CCODE(*i*+1). To activate or inactivate a pair [CCODE(*i*), CCODE(*i*+1)] in which $kind_i$ = 3 or 4, place the same code 'A' or 'N' in both $active_i$ and $active_{i+1}$.

$relop_i$ = '=', '<', or '>' denotes a constraint equation with '=' meaning *equal to*, '<' meaning *less than or equal to*, and '>' meaning *greater than or equal to*. $relop_i$ = '~' denotes a least-squares equation. For a least-squares equation the value SD(*i*) will be used as the a priori standard deviation.

$kind_i$ = 1 specifies an equation of the form

$$f^{(di)}(X(i))\ relop_i\ Y(i)$$

where $f^{(di)}$ denotes the derivative of order $deriv_i$ of the spline function to be determined. The zeroth order derivative is the function itself.

$kind_i$ = 2 specifies an equation of the form

$$f^{(di)}(X(i)) - f^{(di)}(Y(i))\ relop_i\ 0.$$

Note that Y(*i*) is an independent variable value in this case.

$kind_i$ = 3 uses items indexed by both *i* and *i*+1 and specifies an equation of the form

$$f^{(di)}(X(i)) - Y(i+1) \times f^{(di+1)}(X(i+1))\ relop_i\ Y(i)$$

where $f^{(di+1)}$ denotes the derivative of order $deriv_{i+1}$ of the spline function to be determined.

$kind_i$ = 4 uses items indexed by both *i* and *i*+1 and specifies an equation of the form

(The integral from X(*i*) to X(*i*+1) of *f*) $relop_i$ Y(*i*)

See Section C for discussion of expected applications of these different equation forms.

X(), Y() [in] Data for use in building constraint or fitting equations as specified by the contents of CCODE().

SD() [in] SD(*i*) specifies the a priori standard deviation of the error in the equation specified by CCODE(*i*) when $relop_i$ = '~'. The weighted fitting algorithm

will take account of these SD(*i*) values. Optionally, the user may set SD(1) to a negative value. Then this subroutine will use |SD(1)| as the standard deviation for the right-side value in each fitting equation. In this latter case the SD() array can be dimensioned SD(1). Note that a negative value in SD(1) will always be interpreted in this way regardless of the contents of CCODE(1). An error condition is reported if SD(1) = 0.

KORDER [in] Order of the spline function. Each polynomial piece will be of degree at most KORDER-1. The default order of continuity at each internal knot will be KORDER-2. The popular case of a cubic spline with C^2 continuity at the knots is selected by setting KORDER = 4. Require KORDER ≥ 1. Internal arrays in this package impose an upper limit of *kmax* = 20 on KORDER.

NCOEF [in] Number of terms in the sum representing the spline function.

TKNOTS() [in] The knots, t_i, i = 1, ..., NCOEF+KORDER. The interval [t_{KORDER}, $t_{NCOEF+1}$] will be the *proper interpolation interval* for the problem. This interval should contain all the abcissa values occurring in the least-squares and constraint equations, so it is reasonable to set t_{KORDER} less than or equal to the minimum of these abcissae and $t_{NCOEF+1}$ greater than or equal to the maximum. It is convenient and reasonable to set the KORDER-1 knots with indices less than KORDER equal to t_{KORDER}, and the KORDER-1 knots with indices greater than NCOEF+1 equal to $t_{NCOEF+1}$.

Knots indexed from KORDER+1 through NCOEF are *internal* knots. Internal knots specify abcissae at which one polynomial piece ends and the next begins. Successively indexed internal knots may have the same value. A knot appearing with multiplicity μ means the order of continuity of the spline at this knot will be at least KORDER-μ-1. Require 1 ≤ μ ≤ KORDER.

Require $t_i \le t_{i+1}$ for i = 1, ..., NCOEF+KORDER-1; $t_i < t_{i+KORDER}$ for i = 1, ..., NCOEF; $t_{KORDER} < t_{KORDER+1}$; and $t_{NCOEF} < t_{NCOEF+1}$. See Sections C and D for further discussion of knot placement.

BCOEF() [out] Coefficients c_i, i = 1, NCOEF, in the sum representing the spline function as a sum of coefficients times B-spline basis functions.

RNORM [out] Set by the subroutine as a measure of the residual error of the fit.

$$\text{RNORM} := \left[\sum_i 3 \left(\frac{\text{resid}_i}{\text{SD}_i} \right)^2 \right]^{1/2},$$

where the summation is over indices for which $relop_i$ = '~', and $resid_i$ denotes the residual after the fit in the equation specified by CCODE(*i*). Here SD_i denotes SD(*i*) if SD(1) > 0, and |SD(1)| otherwise.

ISET() [in] Array of length 3. These specifications use the following values:

ns = the number of elements in CCODE() containing *relop* = '<' or '>' and *active* = 'A'.
m1 = the number of elements in CCODE() containing *relop* = '=', '<' or '>' and *active* = 'A'.
mfit = the number of elements in CCODE() containing *relop* = '~' and *active* = 'A'.
ntot = NCOEF + *ns*, *mtot* = *m1* + *mfit*, *minmn* = min(*mtot*, *ntot*).

ISET(1) = *ninfo*, the dimension of INFO(). A sufficiently large value is $7 + 2ntot$.
ISET(2) = *nwork*, the dimension of WORK(). A sufficiently large value is
$nwork = mtot \times ntot + 3mtot + 6ntot + 3minmn + m1$.
ISET(3) = *kprint*, a print flag in the range [0, 4]. It is passed on to DBLSE. Zero means no printing. Larger values produce more printing.

INFO() [out and scratch] The first 7 elements of INFO() are used to return information about the problem. The following 2*(NCOEF+*ns*) locations are used as scratch. The dimension of INFO() is *ninfo* given in ISET(1).

INFO(1) = *ierr5*, a status indicator incorporating information from *ierr4* issued by DBLSE. Possible values of *ierr5* are as follows:

ierr5 = 0 means no errors detected.
= 100 means NCOEF < 1
= 150 means KORDER > *kmax* (= 20)
= 200 means TKNOTS(*i*) > TKNOTS(*i*+1)
= 250 means TKNOTS(*i*) ≥ TKNOTS(*i*+KORDER)
= 300 means *ninfo* or *nwork* is too small. Recommended values are returned in INFO(2) and INFO(3).
= 400 means SD(*i*) ≤ 0.0 for some *i*.
= 500 means $deriv_i$ has bad value for some *i*.
= 600 means $relop_i$ has bad value for some *i*.
= 700 means $kind_i$ has bad value for some *i*.
= 800 means $active_i$ has bad value for some *i*.
= 1000 + *ierr4* means *ierr4* ≠ 0 due to error detected in DBLSE.

INFO(2) = *need1*, the dimension needed for INFO().
INFO(3) = *need2*, the dimension needed for WORK().
INFO(4) = *m1*, the number of constraint rows in the matrix representation of the problem. This will be the number of elements in CCODE() containing *relop* = '=', '<' or '>' and *active* = 'A'
INFO(5) = *mfit*, the number of least-squares equations. This will be the number of elements in CCODE() containing *relop* = '~' and *active* = 'A'.
INFO(6) = *ns*, the number of slack variables. This will be the number of elements in CCODE() containing *relop* = '<' or '>' and *active* = 'A'.
INFO(7) = *nsets*, the number of variables in Set *S* at termination. These variables are at values determined by solution of a system of equations. The other NCOEF+*ns-nsets* variables will be at fixed values, either at one of their bounds or at zero.

WORK() [scratch] Work space dimensioned *nwork*. See ISET(2) above.

B3. Usage of DSVAL for Evaluation Using the B-spline Basis

DSVAL returns the value at argument X of the derivative of order IDERIV of the spline function defined by the parameter sequence [KORDER, NCOEF, TKNOTS(), BCOEF()].

Program Prototype, Double Precision

```
INTEGER             NCOEF, KORDER, IDERIV
DOUBLE PRECISION    TKNOTS(≥NCOEF+KORDER), BCOEF(≥NCOEF), X
```

Assign values to all arguments.

```
D = DSVAL ( KORDER, NCOEF, TKNOTS, BCOEF, X, IDERIV )
```

Argument Definitions

KORDER, NCOEF, TKNOTS(), BCOEF() [in] Quantities defining a spline function relative to the B-spline basis as returned by DSFIT or DSFITC. Internal arrays in this subprogram impose an upper limit of *kmax* = 20 on KORDER.

X [in] Argument at which the IDERIV order derivative of the spline function will be evaluated.

IDERIV [in] Derivative order desired. Require IDERIV ≥ 0. Zero means to evaluate the spline function itself.

B4. Usage of DSQUAD for Integration Using the B-spline Basis

DSQUAD returns the value of the integral from X1 to X2 of the spline function defined by the parameter sequence [KORDER, NCOEF, TKNOTS(), BCOEF()].

Program Prototype, Double Precision

```
INTEGER           KORDER, NCOEF
DOUBLE PRECISION  TKNOTS(≥NCOEF+KORDER), BCOEF(≥NCOEF), X1,
X2
```

Assign values to all arguments.

```
D = DSQUAD(KORDER, NCOEF, TKNOTS, BCOEF, X1, X2)
```

Argument Definitions

KORDER, NCOEF, TKNOTS(), BCOEF() [in] Quantities defining a spline function relative to the B-spline basis as returned by DSFIT or DSFITC.

X1, X2 [in] Limits of the integral to be evaluated. Permit X1 < X2 or X1 ≥ X2.

B5. Usage of DSTOP to Convert from the B-spline Basis to the Power Basis

DSTOP converts the representation of a spline function from the B-spline parameterization [KORDER, NCOEF, TKNOTS(), BCOEF()], to the Power basis form [KORDER, NPC, XI(), PCOEF()]. KORDER will not be changed. Typically the B-spline parameters will have come from DSFIT or DSFITC. The Power coefficients can be used by DPVAL and DPQUAD.

Program Prototype, Double Precision

```
INTEGER           KORDER, NCOEF, NPC
DOUBLE PRECISION  TKNOTS(≥NCOEF+KORDER), BCOEF(≥NCOEF),
                  BDIF(≥NCOEF*KORDER), XI(mpc+1),
                  PCOEF(≥KORDER*mpc)
```

The dimension *mpc* must be as large as the output value NPC. In terms of input quantities, it suffices to set *mpc* ≥ NCOEF-KORDER+1. Assign values to KORDER, NCOEF, TKNOTS(), and BCOEF().

```
CALL DSTOP(KORDER, NCOEF, TKNOTS, BCOEF, BDIF, NPC, XI, PCOEF)
```

Results are returned in NPC, XI(), and PCOEF().

Argument Definitions

KORDER, NCOEF, TKNOTS(), BCOEF() [in] Quantities defining a spline function relative to the B-spline basis, as returned by DSFIT or DSFITC.

BDIF() [scratch] Work space of size NCOEF*KORDER.

NPC [out] NPC+1 will be the number of distinct values in the sequence {TKNOTS(*i*), *i* = KORDER, ..., NCOEF+1}. NPC will satisfy NPC ≤ NCOEF-KORDER+1.

XI() [out] A strictly increasing sequence of length NPC+1 consisting of all the distinct values from the sequence {TKNOTS(*i*), *i* = KORDER, ..., NCOEF+1}.

PCOEF() [out] PCOEF(*i*+(*j*-1)*KORDER) will be be the coefficient of (*t* - XI(*j*))**(*i*-1) in the power basis representation of the spline function. (*i* = 1, ..., KORDER; *j* = 1, ..., NPC)

B6. Usage of DPVAL for Evaluation Using the Power Basis

DPVAL returns the value at argument X of the derivative of order IDERIV of the spline function defined by the parameter sequence [KORDER, NPC, XI(), PCOEF()].

Program Prototype, Double Precision

```
INTEGER            NPC, KORDER, IDERIV
DOUBLE PRECISION   XI(≥NPC+KORDER), PCOEF(≥KORDER*NPC), X
```

Assign values to all arguments.

```
D = DPVAL ( KORDER, NPC, XI, PCOEF, X, IDERIV )
```

Argument Definitions

KORDER, NPC, XI(), PCOEF() [in] Quantities defining a spline function relative to the power basis, as returned by DSTOP.

X [in] Argument at which the IDERIV order derivative of the spline function will be evaluated.

IDERIV [in] Derivative order desired. Require IDERIV ≥ 0. Zero means to evaluate the spline function itself.

B7. Usage of DPQUAD for Integration Using the Power Basis

DSQUAD returns the value of the integral from X1 to X2 of the spline function defined by the parameter sequence [KORDER, NPC, XI(), PCOEF()].

Program Prototype, Double Precision

```
INTEGER            KORDER, NPC
DOUBLE PRECISION   XI(≥NPC+KORDER), PCOEF(≥KORDER*NPC), X1, X2
```

Assign values to all arguments.

```
D = DPQUAD(KORDER, NPC, XI, PCOEF, X1, X2)
```

Argument Definitions

KORDER, NPC, XI(), PCOEF() [in] Quantities defining a spline function relative to the power basis, as returned by DSTOP.

X1, X2 [in] Limits of the integral to be evaluated. Permit X1 < X2 or X1 ≥ X2.

B8. Modifications for Single Precision

For single precision usage change the DOUBLE PRECISION statements to REAL and change the initial "D" in the subprogram names to "S".

C. Examples and Remarks

Demonstration of DSFIT. The demonstration driver DRDSFIT sets up a curve fitting problem having 12 (x,y) pairs of data. It uses DSFIT to do a least-squares fit to this data with an 8-parameter cubic spline function. It uses DSVAL to evaluate the fitted function over the given set of x values. It uses DSQUAD to compute the definite integral of the fitted function from 5.0 to 20.0. It uses DSTOP to convert the B-spline representation to the power representation. It then uses DPVAL and DPQUAD to repeat the function evaluation and integral computation using the power representation. The output is listed in ODDSFIT.

Demonstration of DSFITC. The program DRDSFITC illustrates the use of DSFITC to compute a constrained least-squares spline fit to data. Output from DRDSFITC is listed in ODDSFITC. We have 24 data points, given in the the first 24 entries of the arrays XI() and YI() in the DATA statement in DRDSFITC.

Suppose these data are measurements of some phenomenon that is known to be monotone nondecreasing and we wish to find a monotone nondecreasing function that closely fits the data. An unconstrained least-squares fit to this data by a single polynomial or by a polynomial spline function will have unwanted oscillations. There are rational functions and exponential functions with three parameters that are monotone and of somewhat the desired shape, but these functions do not have enough free parameters to allow the function to fit the data really closely. A satisfactory fit can be obtained using a cubic spline function having C^2 continuity.

The data abcissae range from 0 to 6. We shall place quadruple knots at these two points and internal knots at 1.5, 2.5, 3.3, 4.0, and 4.7. The number and locations of these internal knots were selected by some trial and error. These knot values are stored in TKNOTS(). Since we have selected a total of 13 knots and we have KORDER = 4 to specify a cubic spline, the number of coefficients will be set to NCOEF = 13 - 4 = 9.

We shall require the curve to be concave up over [0, 2.5] by requiring $f'' \geq 0$ at 0 and at the first two internal knots. These constraints, along with the constraint $f'(0) \geq 0$, will force f' to be nonnegative over [0, 2.5]. Similarly we require the curve to be concave down over [3.3, 6] by requiring $f'' \leq 0$ at the last three internal knots and at 6. These constraints, along with the constraint $f'(6) \geq 0$, will force f' to be nonnegative over [3.3, 6]. It follows that $f' \geq 0$ over [2.5, 3.3] since in this interval f' is a quadratic polynomial that is nonnegative and nondecreasing at 2.5 and nonnegative and nonincreasing at 3.3.

The second derivative of a cubic spline is linear between knots. Thus in the interval between the successive knots at 2.5 and 3.3, the conditions $f''(2.5) \geq 0$ and $f''(3.3) \leq 0$ imply that f'' can have at most one sign change in this interval. This assures the only inflection point of the curve over (0, 6) will occur in the interval [2.5, 3.3].

Supposing we also wish to have the fitted curve take the value 1 at 0 and the value 5 at 6, we also impose these constraints.

Recall that the four characters in each entry of CCODE() are interpreted as (*kind*, *deriv*, *relop*, *active*). All but the last element of CCODE() have *active* = 'a', meaning these elements are active, while the last element has *active* = '!', which is the termination signal. All of the active elements have *kind* = '1', meaning the specified equation is of the form,

$$f^{(di)}(\mathrm{XI}(i))\ relop_i\ \mathrm{YI}(i).$$

The first 24 entries have *relop* = '~', meaning that each specifies one of the least-squares equations. The ten elements of CCODE() beginning with CCODE(25) specify the constraints. For example CCODE(26) has *deriv* = '1' and *relop* = '>' meaning the first derivative at XI(26) is constrained to be ≥ YI(26).

Using Constraints to Control Shape. A function with at least C^1 continuity is nondecreasing over [*c*,*d*] if its first derivative is nonnegative throughout [*c*,*d*]. A function with at least C^2 continuity is concave up over an interval [*c*,*d*] if its second derivative is nonnegative throughout [*c*,*d*]. Although these are properties defined over an interval, it is possible to impose these conditions on spline functions by appropriate assignment of constraints at a finite number of points.

Note that if *f* is a cubic spline function with C^2 continuity, then *f*' is a quadratic spline with C^1 continuity, and *f*" is a linear spline function with C^0 continuity, i.e., *f*" is a continuous piecewise linear function with possible slope changes only at knots. It follows that by requiring *f*" ≥ 0 at *c* and *d*, and at any knots between *c* and *d*, *f*" will necessarily be nonnegative throughout [*c*,*d*], and therefore *f* will be concave up throughout [*c*,*d*].

If one wants *f* to be monotone nondecreasing as well as concave up over [*c*,*d*] it suffices to require *f*'(*c*) ≥ 0 along with the second derivative conditions already discussed, since with *f*" ≥ 0 throughout [*c*,*d*], *f*' cannot have a smaller value anywhere in [*c*,*d*] than it has at *c*.

If one wants monotonicity for *f* without second derivative constraints, one could let *f* be a quadratic spline rather than a cubic spline. Then *f*' will be piecewise linear and one can control the sign of *f*' over an interval by constraining *f*' at knots as was done above for *f*". If one prefers to use cubic splines, one can do trial and error placement of constraints on *f*' and eventually keep *f*' from changing sign.

Periodicity. When periodicity is desired it should be specified for the function value and all orders of derivatives that are continuous at the points referenced in the specification. For example, suppose one wants periodicity of 360 degrees. The *proper interpolation interval* could be set as [*a*,*b*] = [0.0, 360.0]. If one uses KORDER = 4 one should specify periodicity for *f*, *f*', and *f*". This can be done by setting CCODE(1:3) := '20=a', '21=a', '22=a'; X(1:3) := 0.0, 0.0, 0.0; and Y(1:3) := 360.0, 360.0, 360.0.

Although it is not essential to do so, one may wish to have the periodicity reflected in the coefficients. Letting *p* denote the period, and assuming $p+t_{\mathrm{KORDER}} = p+a = b = t_{\mathrm{NCOEF+1}}$, this can be done by setting the initial knots as $t_i := -p+t_{\mathrm{NCOEF+1-KORDER+i}}$, for i = 1, ..., KORDER-1, and the final knots as $t_{\mathrm{NCOEF+1+i}} := p+t_{\mathrm{KORDER+i}}$, for i = 1, ..., KORDER-1. Then the coefficients will reflect the periodicity by satisfying $c_{\mathrm{NCOEF+1-KORDER+i}} = c_i$, for i = 1, ..., KORDER-1.

Differential Equations. Using *kind* = 3, conditions such as *f*'(*x*) - *cf*(*x*) = *d*, or *f*'(*x*) - *cf*(*x*) ~ *d* (and slightly more general expressions) can be specified for given values of *x*, *c* and

d. Thus DSFITC can be used to implement the collocation method of computing an approximate solution to linear differential equations.

Assignment of Knots. Many discussions of spline interpolation are based on the assumption that many, or all, of the knots will be assigned to coincide with data abscissas. This is not necessary, either for interpolation or least-squares fitting.

The necessary and sufficient condition for an unconstrained interpolation or least-squares fitting problem using a family of NC B-spline basis functions of order K to give rise to a full-rank matrix, and thus have a unique solution, is that there are at least NC distinct data abscissas, and that it is possible to choose NC distinct data abscissas and relabel them, say as u_i, i = 1, NC, so they will satisfy $B_i(u_i) \neq 0$ for i = 1, ..., NC. This condition will be satisfied if the (possibly relabled) u_i's relate to the knots according to $t_i < u_i < t_{i+K}$ for i = 1, ..., NC.

Consider an interpolation problem with NXY data points, all data abscissas being distinct. Choose a spline order K ≥ 2. We must use exactly NC = NXY B-spline basis functions. Thus NC+K knots must be assigned. Let *a* and *b* be the minimum and maximum data abscissas respectively. Assign the first K knots the value *a* and the last K knots the value *b*. Then NC-K knots remain to be assigned and there are NC-2 data abscissas distinct from *a* and *b*. One simple approach is to use any NC-K of these NC-2 data abscissas as knots. In the popular case of cubic spline interpolation (K = 4), there would be just two data abscissas not used as knots. It is common to choose these to be the first one after *a* and the last one before *b*.

Another method suggested in Ref 1, pp. 218-219, for assigning the interior knots for interpolation is the formula

$$t_i = (u_{i-K+1} + ... + u_{i-1}) / (K-1), \quad i = K+1, ..., NC$$

where the ordered set of data abscissas are denoted by u_i, i = 1, ..., NC.

For least-squares fitting one must choose NC < NXY.

D. Functional Description

Representation of an individual B-spline basis function. Let $\{t_1, ..., t_{K+1}\}$ be a set of strictly increasing real numbers we will call *knots*. To within a multiplicative scale factor, there is one, and only one, spline polynomial function of order K (i.e., having polynomial pieces of degree at most K-1) having at least C^{K-2} continuity at these knots, and being nonzero throughout the open interval (t_1, t_{K+1}), and zero outside this interval. With some convention for assigning the scale factor, such a function is called a *B-spline basis function*. The interval $\langle t_1, t_{K+1} \rangle$ will be called the *support* interval for this B-spline basis function. We use angle brackets $\langle \rangle$ to indicate that we are not specifying whether the endpoints are included or not.

Following Ref 1, these definitions can be generalized to allow knots to coelesce. This has a natural mathematical interpretation of reducing the order of continuity at the affected knots. At an ordinary non-multiple knot, a spline function of order K has C^{K-2} continuity. At a knot of multiplicity, μ a spline function of order K has $C^{K-\mu-1}$ continuity. For example, a B-spline basis function of order 4 defined over the knot set {0, 1, 1, 5, 6} consists of only three nontrivial cubic pieces, and these have C^2 continuity at 0, 5, and 6, but only C^1 continuity at 1. This freedom to lower the order of continuity at specified places can be useful in allowing a curve to change direction more sharply at such a point.

Representation of a Spline Function Using the B-Spline Basis. Suppose we wish to construct a family of spline functions of order K, and having NC degrees of freedom, over an interval $[a,b]$, which will be called the *proper interpolation interval* for this spline family. We require $a < b$ and NC ≥ K. Construct a *knot sequence* $T = \{t_1, ..., t_{NC+K}\}$. This sequence must be nondecreasing and have $t_K = a$ and $t_{NC+1} = b$. The values of the knots indexed before K or after NC+1 do not affect the shapes that can be achieved by the family of splines to be defined. A convenient way to set these knots is to set $t_1 = ... = t_K = a$, and $t_{NC+1} = ... = t_{NC+K} = b$.

The knots indexed from K+1 through NC are called *interior knots.* Internal knots define where the different polynomial pieces meet. Their placement determines the shapes that the resulting spline family can achieve. We require $t_K < t_{K+1}$, and $t_{NC} < t_{NC+1}$, and $t_i < t_{i+K}$ for $i = 1, ..., NC$. Within the limitations of these constraints, successive interior knots need not be distinct. At a knot of multiplicity μ members of this spline family will have $C^{K-\mu-1}$ continuity.

For each $i = 1, ..., NC$, we associate a B-spline basis function B_i having t_i, t_{i+K} as its support interval. For $i = K, ..., NC$, each interval t_i, t_{i+1} is in the support interval of exactly K basis functions, namely B_j, for $j = i-K+1, ..., i$. If such an interval t_i, t_{i+1} has nonzero length, the K basis functions that contain this interval in their support intervals form a basis for the space of all polynomials of degree ≤ K-1 over this interval. The closed union of these intervals is the interval $[a,b]$. This is the interval over which it is most reasonable to use linear combinations of the B_i's to fit data.

Any polynomial of degree ≤ K-1 can be exactly represented over $[a,b]$ by a linear combination of the B_i's, $i = 1, ..., NC$. In particular, the constant function whose value is one is representable over $[a,b]$ by a linear combination of the B_i's. In this package the scaling of the B_i's is determined by the requirement that all the coefficients in this linear combination be ones, i.e., $\sum_{i=1}^{NC} B_i(t) = 1$ for all $t \in [a,b]$.

Given coefficients, c_i, i = 1, ..., NC, a spline function, $f(t)$, is represented for $t \in [a,b]$ as $f(t) = \sum_{i=1}^{NC} c_i B_i(t)$. Although this is a sum of NC terms, at most K of the terms are nonzero at any single point, t, due to the properties of the basis functions. For evaluation of $f(t)$ at a point $t \in (a,b)$ that coincides with a knot, this package uses the polynomial piece defined over the nonzero subinterval immediately to the right of t. This package allows extrapolation outside the interval (a,b) using the convention that for $t \le a$, the package will extend the polynomial that is defined over t_K, t_{K+1}, and for $t \ge b$ the package will extend the polynomial that is defined over t_{NC}, t_{NC+1}.

Within this package a spline function is fully specified relative to the B-spline basis by two integers, KORDER and NCOEF, and two floating point arrays, TKNOTS() and BCOEF(), containing $\{t_i, i = 1, ..., NCOEF+KORDER\}$ and $\{c_i, i = 1, ..., NCOEF\}$.

Representation of a Spline Function Using the Piecewise Power Basis. Assume a spline function $f(t)$ has been defined relative to the B-spline basis as described above. Let NPC be the number of subintervals of nonzero length into which $[a,b]$ is partitioned by the knot sequence T. Let $x_j, j = 1, ..., NPC$ be the left endpoints of these subintervals, and let $x_{NPC+1} = b$. For the half-open subinterval $[x_j, x_{j+1})$ coefficients $p_{i,j}$ can be determined so the polynomial

$$p_{1,j} + p_{2,j}h + p_{3,j}h^2 + ... + p_{K,j}h^{K-1}$$

with $h = (t-x_j)$, is identical to the polynomial spline function $f(t)$ over this interval. If evaluation for t outside $[a,b)$ is requested, the package will use the coefficients indexed by $j = 1$ if $t < a$, and will use $j =$ NPC if $t \geq b$.

Within this package a piecewise polynomial represented relative to the *power* basis is specified by two integers, KORDER and NPC, and two floating point arrays, XI() and PCOEF(), containing $\{x_i, i = 1, ..., NPC+1\}$ and $\{p_{i,j}, i = 1, ..., KORDER; j = 1, ..., NPC\}$.

The power representation does not inherently assure any particular order of continuity at the knots. However, if the coefficients are determined by conversion from a B-spline representation, they will represent the same spline function and thus have the same continuity properties.

Computation Using B-Spline Basis Functions Suppose a spline function f is defined relative to a B-spline basis by the quantities KORDER, NCOEF, T() and BCOEF() as discussed above. The *proper interpolation interval* for f is $[a,b]$ where $a = t_{KORDER}$, and $b = t_{NCOEF+1}$. With any argument x we associate a *reference index*, j, and *reference interval* t_j, t_{j+1} having $t_j < t_{j+1}$. If $x \in [a,b)$, j is chosen so that $x \in [t_j,t_{j+1})$, otherwise, if $x < a$ set $j :=$ KORDER, and if $x \geq b$ set $j :=$ NCOEF. Given x, subprogram DSFIND determines its reference index. From an initial trial value for j the subprogram searches forward or backward, doubling the index increment for each trial, until either a bracketing pair of knots is found or the search reaches one end of the specified search range. If a bracketing interval is found, bisection is used, if necessary, to reduce the interval to the prescribed form.

To describe the computational algorithms, we need to consider families of lower order basis functions over the same knot sequence T. Let NT denote the number of knots in T, i.e., NT = KORDER+NCOEF. For $k = 1, ..., KORDER$, let $\{B_{i,k}, i = 1, ..., NT-k\}$, be the set of B-spline basis functions of order k associated with T. The support interval for the function $B_{i,k}$ is t_i, t_{i+k}. Formally one may follow Ref 1, p. 118, and define $B_{i,k} \equiv 0$ if $t_i = t_{i+k}$, however it happens that these functions do not occur in the algorithms we consider.

A B-spline basis function of order k can be expressed in terms of two basis functions of order k-1 as:

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(x) + \frac{t_{i+1} - x}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(x). \quad (1)$$

This formula was discovered and published independently by M. G. Cox and C. de Boor in 1972. It is a very favorable formula with regard to propagation of roundoff error since, except when used for extrapolation, the B's and the factors multiplying the B's are always nonnegative, so the central "+" always involves addition of nonnegative quantities.

Consider now the problem in which we are given an x and its reference index j, and we need to compute values at x of the KORDER basis functions of order KORDER that are nonzero on t_j, t_{j+1}. These functions will be $B_{j-KORDER+1,KORDER}$ through $B_{j,KORDER}$.

Among the basis functions of order 1, only $B_{j,1}$ is nonzero on t_j, t_{j+1} and its value is 1 throughout this interval. At order 2 only the two basis functions $B_{j-1,2}$ and $B_{j,2}$ are nonzero on t_j, t_{j+1}. These can be computed using Eq(1) and the known values of $B_{j-1,1}$, $B_{j,1}$, and $B_{j+1,1}$, which are 0, 1, and 0, respectively. Clearly this process can be continued until the values of the KORDER nonzero basis functions of order KORDER are computed. This method is implemented in subprogram DSBASD. For the case of cubic

splines (KORDER = 4), this involves nine applications of Eq(1) and in six of these applications one of the entering B's is known to be zero.

To evaluate a spline function f at a given argument x, one could use this method to evaluate the nonzero basis functions and then form the sum of these multiplied by the coefficients that define f. There is a more efficient method however. For a spline function f of order k, its evaluation for a point x with reference index j can be expressed as $f(x) = \sum_{i=j-k+1}^{j} c_{i,k} B_{i,k}(x)$. Replacing each $B_{i,k}$ in this expression by the right side of Eq(1) and collecting terms on the $B_{i,k-1}$'s, and noting that only k-1 of these (k-1)-order basis functions are nonzero on the reference interval, gives the expression $f(x) = \sum_{i=j-k+2}^{j} c_{i,k-1}(x) B_{i,k-1}(x)$ where

$$c_{i,k-1}(x) = \frac{(x - t_i)c_{i,k} + (t_{i+k-1} - x)c_{i-1,k}}{t_{i+k-1} - t_i}. \qquad (2)$$

One can continue reducing the spline order and the number of terms in the sum in this way, finally reaching spline order 1 with only one term in the sum: $f(x) = c_{j,1}(x)B_{j,1}(x) \equiv c_{j,1}(x)$, since $B_{j,1}(x) = 1$.

Thus, as an algorithm for evaluating a spline function of order KORDER at an argument x with reference index j, one initializes the process by setting $c_{i,\mathrm{KORDER}} := c_i$, for $i = j$-KORDER+1, ..., j. Then for k = KORDER, KORDER-1, ..., 2, one computes $c_{i,k-1}$ for $i = j$-k+2, ..., j, using Eq(2). The final quantity $c_{j,1}$ is the value $f(x)$. This method is implemented in subprogram DSVAL. For the case of cubic splines (KORDER = 4), this involves six applications of Eq(2).

The first derivative of a spline function f of order k is a spline function of order k-1 over the same knot sequence. For an x with reference index j we have $f(x) = \sum_{i=j-k+1}^{j} c_{i,k} B_{i,k}(x)$ and $f(x) = \sum_{i=j-k+2}^{j} c^{(1)}_{i,k-1} B_{i,k-1}(x)$, where

$$c^{(1)}_{i,k-1} = \frac{(k-1)(c_{i,k} - c_{i-1,k})}{t_{i+k-1} - t_i}. \qquad (3)$$

To compute the value of the n^{th} derivative of a spline function, Eq(3) can be applied as many times as necessary to compute coefficients of a B-spline representation of the n^{th} derivative, and then Eq(2) can be used to evaluate the derivative. This algorithm is implemented in DSVAL with all computation being done from scratch for a given x.

Since Eq(3) does not involve x, it is possible to use Eq(3) to precompute an array of coefficients for later use in computing derivative values for many x values. This approach is implemented in DSDIF and DSVALA. DSDIF computes the array of coefficients for all derivatives of orders up to a specified NDERIV and DSVALA uses these coefficients in computing the values of all derivatives of orders up to NDERIV for a given x. Subprogram DSTOP for the conversion from the B-spline basis to the power basis uses DSVALA, since the coefficents relative to the power basis are just derivatives of the spline function divided by factorials.

For an x with reference index j, DSBASD computes the NDERIVth derivative of the KORDER basis functions of order KORDER that are nonzero on the reference interval. From Eq(3) we can express the first derivative of a single basis function as

$$B'_{i,k}(x) = (k-1)\left[\frac{B_{i,k-1}(x)}{t_{i+k-1}-t_i} - \frac{B_{i+1,k-1}(x)}{t_{i+k}-t_{i+1}}\right]. \tag{4}$$

DSBASD first uses Eq(1) to compute the values $B_{i,KORDER-NDERIV}(x)$, for i = j-KORDER+NDERIV+1, ..., j. Then for d = 1, ..., NDERIV, DSBASD uses Eq(4) to compute $B^{(d)}_{i,KORDER-NDERV+d}(x)$, for i = j-KORDER+NDERIV-d+1, ..., j.

It is possible to derive formulas for the exact integration of a spline function by appropriate inverse use of Eq(3). The resulting method is unwieldy, and suspect with regard to propagation of roundoff error. Instead we follow the approach of Amos, Ref 2, that uses Gaussian quadrature. An n-point Gaussian quadrature formula is exact for polynomials up to degree 2n-1. The formula is applied separately to each polynomial piece needed to cover a specified integration interval. This method is used in DSQUAD and in DSBASI. Each of these subprograms contains stored constants for 2, 6, and 10-point Gaussian formulas. The 2-point formula is used for KORDER from 1 to 4, the 6-point formula from 5 to 12, and the 10-point formula from 13 to 20.

The fitting subroutine DSFIT uses DSBASD to form rows of the matrix for the least-squares problem. Each row will have at most KORDER nonzeros in consecutive locations giving rise to a block-banded form for the matrix. DSFIT uses DBACC and DBSOL to process and solve this system. This approach takes advantage of both the band structure and sequential processing to reduce the amount of working space needed.

The constrained fitting subroutine DSFITC uses DSBASD and DSBASI, as appropriate to form rows of matrices representing the constraint conditions and the least-squares problem. Due to the general form of constraints allowed, the overall problem is not assumed to have a banded form so the matrices are formed in full. The resulting problem is linear least-squares with general linear equality and inequality constraints which is solved using DBLSE.

References

1. Carl de Boor, *A Practical Guide to Splines*, Springer-Verlag, 1978.
2. D. E. Amos, Sandia report SAND79-1825, Sandia Laboratory, Albuquerque, June, 1979.
3. T. M. Lang, R. J. Hanson, and D. R. Campbell, *French Curve*, JPL internal computing memorandum No. 203, Sept 1968, revised Sept 1969, 27 pp.

E. Error Procedures and Restrictions

DSFIT, DSFITC, and DSVAL each contain an internal dimensioning parameter *kmax* = 20. It is an error if KORDER > *kmax* in any of these subprograms.

DSFIT handles any detected error by setting IERR1, reporting the error to the library error message processing subroutines of Section 19.2 and then returning. DSFITC handles errors similarly, setting the indicator *ierr5* in INFO(1).

The only error detected in DSVAL is KORDER > *kmax*, in which case DSVAL calls the library error processing subroutines with severity level 2 which nominally causes message printing and termination of execution.

Abcissae and weights for 2-point, 6-point, and 10-point Gaussian quadrature are stored to 40 decimal digits in DSQUAD. With infinite precision abcissae and weights, these formulae would be exact for splines of KORDER up to 20. DPQUAD does not use any inexact stored constants.

F. Supporting Information

Subroutines DSFIT and DSFITC were written in the JPL SFTRAN structured Fortran language and transformed to Fortran 77. The other subprograms were written directly in Fortran77.

Program Unit and Entry Name	**External References**
DSFIT	DBACC, DBSOL, DERV1, DSBASD, ERMSG, IERM1, IERV1
DSFITC	DBLSE, DERV1, DSBASD, DSBASI, DSFIND, ERMOR, ERMSG, IERM1, IERV1
DSVAL	DSFIND, IERM1, IERV1
DSQUAD	DSFIND, DSVAL
DSTOP	DSDIF, DSVALA
DPVAL	DSFIND
DPQUAD	DSFIND
SSFIT	SBACC, SBSOL, SERV1, SSBASD, ERMSG, IERM1, IERV1
SSFITC	SBLSE, SERV1, SSBASD, SSBASI, SSFIND, ERMOR, ERMSG, IERM1, IERV1
SSVAL	SSFIND, IERM1, IERV1
SSQUAD	SSFIND, SSVAL
SSTOP	SSDIF, SSVALA
SPVAL	SSFIND
SPQUAD	SSFIND

DRDSFIT

```
c     program DRDSFIT
c>> 1992-10-29 C. L. Lawson, JPL
c    Demonstration driver for DSFIT, DSVAL, DSQUAD, DSTOP, DPVAL, DPQUAD
c     ------------------------------------------------------------------
c--   D version uses DRDSFIT, DSFIT, DSVAL, DSQUAD, DSTOP, DPVAL, DPQUAD
c--&                 DPRPL
c--   S version uses DRSSFIT, SSFIT, SSVAL, SSQUAD, SSTOP, SPVAL, SPQUAD
c--&                 SPRPL
c     ------------------------------------------------------------------
      integer I, IERR, NXY, KORDER, MPC, NCOEF, NDERIV, NPC, NT, LDW
      parameter(NXY = 12, NCOEF=8, KORDER=4, NT = NCOEF+KORDER,LDW = 10)
      parameter(MPC = NCOEF-KORDER+1)
      double precision BDIF, DSVAL, DSQUAD, DPVAL, DPQUAD
      double precision BCOEF(NCOEF), PCOEF(MPC*KORDER)
      double precision SD(1), SIGFAC, TKNOTS(NT), W(LDW,KORDER+1)
      double precision X(NXY), XI(MPC+1), Y(NXY), YFIT, Z
```

```
      character IMAGE*31
      data X / 2.D0, 4.D0, 6.D0, 8.D0,10.D0,12.D0,
     *        14.D0,16.D0,18.D0,20.D0,22.D0,24.D0/
      data Y /2.2D0,4.0D0,5.0D0,4.6D0,2.8D0,2.7D0,
     *        3.8D0,5.1D0,6.1D0,6.3D0,5.0D0,2.0D0/
      data TKNOTS / 4*2.0D0, 6.4D0, 10.8D0, 15.2D0, 19.6D0, 4*24.0D0 /
      data NDERIV / 0 /
      data SD(1) / -1.0D0 /
c     ----------------------------------------------------------------
      print'(a)',' DRDSFIT',
     * ' Demo driver for DSFIT, DSVAL, DSQUAD, DSTOP, DPVAL, DPQUAD'
      print'(/a,i3,a,i3)','   KORDER =',KORDER,',  NCOEF =',NCOEF
      print'(a,4f10.5/(14x,4f10.5))','   TKNOTS() = ',(TKNOTS(I),I=1,NT)
      call DSFIT(X, Y, SD, NXY, KORDER, NCOEF, TKNOTS, BCOEF,
     *           SIGFAC, IERR, LDW, W)
      print'(/a/a,i5,a, f10.5 //a,4f10.5/(13x,4f10.5))',
     *  ' After call to DSFIT:',
     *  '   IERR =',IERR,',  SIGFAC =',SIGFAC,
     *  '   BCOEF() = ',(BCOEF(I),I=1,NCOEF)

      print'(/a)',' Evaluating fitted spline function using DSVAL:'
      print'(/a/) ',
     *    '    I    X       Y       YFIT  R=Y-YFIT               YFIT'

      do I=1,NXY                     !  Using Fortran 90 "DO" syntax.
       YFIT=  DSVAL(KORDER, NCOEF, TKNOTS, BCOEF, X(I), NDERIV)
       call DPRPL(YFIT, '*', IMAGE, 31, 1.9d0, 6.3d0, .true.)
       print'(3x,i2,f6.0,2f9.3,f10.3,3x,a)',
     *    I, X(I), Y(I), YFIT, Y(I)-YFIT, IMAGE
      enddo ! I

      Z = DSQUAD(KORDER, NCOEF, TKNOTS, BCOEF, 5.0d0, 20.0d0)
      print'(/a,f12.5)',' Integral from 5.0  to 20.0 using DSQUAD:',Z

      call DSTOP(KORDER, NCOEF, TKNOTS, BCOEF, BDIF, NPC, XI, PCOEF)
      print'(/a)',
     * ' Using DSTOP to convert from B-spline basis to power basis.'
      print'(a,i3)','        NPC =',NPC
      print'(a,4f10.5/(14x,4f10.5))','       XI() = ',(XI(I),I=1,NPC+1)
      print'(a,4f10.5/(14x,4f10.5))',
     *   '    PCOEF() = ',(PCOEF(I),I=1,NPC*KORDER)

      print'(/a)',' Evaluating fitted spline function using DPVAL:'
      print'(/a/) ',
     *    '    I    X       Y       YFIT  R=Y-YFIT               YFIT'

      do I=1,NXY                     !  Using Fortran 90 "DO" syntax.
       YFIT=  DPVAL(KORDER, NPC, XI, PCOEF, X(I), NDERIV)
       call DPRPL(YFIT, '*', IMAGE, 31, 1.9d0, 6.3d0, .true.)
       print'(3x,i2,f6.0,2f9.3,f10.3,3x,a)',
     *    I, X(I), Y(I), YFIT, Y(I)-YFIT, IMAGE
```

```
      enddo ! I

      Z = DPQUAD(KORDER, NPC, XI, PCOEF, 5.0d0, 20.0d0)
      print'(/a,f12.5)',' Integral from 5.0  to 20.0 using DPQUAD:',Z
      end ! DRDSFIT
```

ODDSFIT

```
 DRDSFIT
 Demo driver for DSFIT, DSVAL, DSQUAD, DSTOP, DPVAL, DPQUAD

   KORDER =  4,  NCOEF =  8
   TKNOTS() =    2.00000   2.00000   2.00000   2.00000
                 6.40000  10.80000  15.20000  19.60000
                24.00000  24.00000  24.00000  24.00000

 After call to DSFIT:
   IERR =    0,  SIGFAC =   0.14664

   BCOEF() =    2.20672   3.33355   7.10955   0.91845
                4.88398   7.24971   5.03117   1.99475

 Evaluating fitted spline function using DSVAL:

    I    X       Y       YFIT  R=Y-YFIT               YFIT

    1    2.    2.200    2.207    -0.007     *
    2    4.    4.000    3.958     0.042                *
    3    6.    5.000    5.111    -0.111                        *
    4    8.    4.600    4.430     0.170                    *
    5   10.    2.800    2.959    -0.159          *
    6   12.    2.700    2.646     0.054        *
    7   14.    3.800    3.734     0.066                *
    8   16.    5.100    5.162    -0.062                        *
    9   18.    6.100    6.132    -0.032                                *
   10   20.    6.300    6.233     0.067                                 *
   11   22.    5.000    5.033    -0.033                         *
   12   24.    2.000    1.995     0.005    *

 Integral from 5.0  to 20.0 using DSQUAD:    66.54641

 Using DSTOP to convert from B-spline basis to power basis.
        NPC =  5
       XI() =    2.00000   6.40000  10.80000  15.20000
                19.60000  24.00000
    PCOEF() =    2.20672   0.76829   0.11795  -0.03213
                 5.13370  -0.05990  -0.30617   0.04307
                 2.61122  -0.25290   0.26231  -0.02300
                 4.61735   0.71946  -0.04132  -0.00801
                 6.30079  -0.10933  -0.14704  -0.01148

 Evaluating fitted spline function using DPVAL:
```

```
   I    X       Y       YFIT  R=Y-YFIT               YFIT

   1    2.    2.200    2.207    -0.007     *
   2    4.    4.000    3.958     0.042                *
   3    6.    5.000    5.111    -0.111                      *
   4    8.    4.600    4.430     0.170                   *
   5   10.    2.800    2.959    -0.159          *
   6   12.    2.700    2.646     0.054        *
   7   14.    3.800    3.734     0.066                *
   8   16.    5.100    5.162    -0.062                      *
   9   18.    6.100    6.132    -0.032                               *
  10   20.    6.300    6.233     0.067                                *
  11   22.    5.000    5.033    -0.033                         *
  12   24.    2.000    1.995     0.005    *

Integral from 5.0  to 20.0 using DPQUAD:    66.54641
```

DRDSFITC

```
c     program DRDSFITC
c>> 1993-01-13 C. L. Lawson, JPL
c>> 1992-11-10 C. L. Lawson, JPL
c>> 1992-11-04 C. L. Lawson, JPL
c>> 1989-03-02 C. L. Lawson, JPL
c>> 1988-04-01 C. L. Lawson, JPL
c     DRDSFITC..  Demo driver for DSFITC, Spline fit with constraints.
c     The problem has 24 data points and 10 constraints.
c     The spline is order 4 with 9 coefficients.
c     ------------------------------------------------------------------
c--   D version uses DRDSFITC, DSFITC, DSVAL, DPRPL, dble
c--   S version uses DRSSFITC, SSFITC, SSVAL, SPRPL, real
c     ------------------------------------------------------------------
      integer I, KORDER, KPRINT, MXY, MT
      integer NCOEF, NDATA, NINFO, NWORK
      parameter( NDATA = 24, MXY = NDATA+10)
      parameter( NINFO = 41, NWORK = 843)
      parameter( NCOEF=9, KORDER = 4, MT = NCOEF+KORDER)
      parameter( KPRINT = 0)
      integer INFO(NINFO), ISET(3)
      double precision DSVAL
      double precision BCOEF(NCOEF), BDIF(NCOEF*3), DELX, RNORM
      double precision SDI(MXY), SMAX, SMIN, SVALUE(0:2)
      double precision TKNOTS(NCOEF+KORDER)
      double precision WORK(NWORK), X, XI(MXY), YI(MXY), YFIT
      character CCODE(MXY+1)*4, IMAGE*49
      data TKNOTS / 4*0.0, 1.5, 2.5, 3.3, 4.0, 4.7, 4*6.0/
      data CCODE /  24*'10~a',
     *             '10=a', '11>a', '12>a', '12>a', '12>a',
     *             '12<a', '12<a', '12<a', '11>a', '10=a',
     *             '   !'/
      data XI / 0.0, 0.3, 0.7, 1.0, 1.3, 1.7, 2.0, 2.3,
```

```
     *          2.5, 2.6, 2.8, 2.9, 3.0, 3.1, 3.2, 3.5,
     *          3.7, 4.0, 4.3, 4.7, 5.0, 5.3, 5.7, 6.0,
     *          0.0, 0.0, 0.0, 1.5, 2.5,
     *          3.5, 4.5, 6.0, 6.0, 6.0/

      data YI /  1.0,  1.1, 0.9, 1.02,  1.2, 1.0, 1.2, 1.4,
     *          1.76,  2.0, 2.4,  2.6,  3.0, 3.4, 3.7, 4.3,
     *          4.45, 4.76, 4.8,  5.0, 4.96, 4.9, 4.9, 5.0,
     *          1.0, 0.0, 0.0, 0.0, 0.0,
     *          0.0, 0.0, 0.0, 0.0, 5.0/

      data SDI(1) / -1.0D0 /
      data ISET  / NINFO, NWORK, KPRINT /
C     ------------------------------------------------------------------
      print'(1x,a)','DRDSFITC..  Demo driver for DSFITC',
     * 'Least-squares polynomial spline fit to data with constraints.'
C
      print'(/a)',
     *   '   I   kind   deriv   relop  active       X         Y'
      do I = 1,MXY
         print'(1x,i3,3x,a,7x,a,7x,a,7x,a,7x,a,f12.3,f10.3)', I,
     *      CCODE(I)(1:1),CCODE(I)(2:2),CCODE(I)(3:3),CCODE(I)(4:4),
     *      XI(I),YI(I)
      enddo ! I
      I = MXY+1
      print'(1x,i3,3x,a,7x,a,7x,a,7x,a)', I,
     *      CCODE(I)(1:1),CCODE(I)(2:2),CCODE(I)(3:3),CCODE(I)(4:4)
      print'(/a,i3,a,i3)','   KORDER =',KORDER,',  NCOEF =',NCOEF
      print'(a,4f10.5/(14x,4f10.5))','   TKNOTS() = ',(TKNOTS(I),I=1,MT)
C
      call DSFITC(CCODE, XI, YI, SDI, KORDER, NCOEF, TKNOTS,
     *               BCOEF, RNORM, ISET, INFO, WORK)
C
      print'(/a)',' After call to DSFITC:'
      print'(/3x,a,i6,a,i7,a,i7/ 3x,a,i6,a,i7,a,i7/ 3x,a,f12.5)',
     * 'IERR5 =',INFO(1),',   NEED1 =',INFO(2),',   NEED2 =',INFO(3),
     * 'M1    =',INFO(4),',   MFIT  =',INFO(5),',   NS    =',INFO(6),
     * 'RNORM =',RNORM
      print'(/a,4f10.5/(13x,4f10.5))',
     *  '   BCOEF() = ',(BCOEF(I),I=1,NCOEF)
C
      call DSDIF(KORDER, NCOEF, TKNOTS, BCOEF, 2, BDIF)
      SMIN = 0.0d0
      SMAX = 0.0d0
      DELX = (XI(NDATA)-XI(1))/30.0D0
      X = XI(1)
      do I=0,31                 !  Using Fortran 90 "DO" syntax.
         call DSVALA(KORDER, NCOEF, TKNOTS, 2, BDIF, X, SVALUE)
         SMIN = min(SMIN, SVALUE(0), SVALUE(1), SVALUE(2))
         SMAX = max(SMAX, SVALUE(0), SVALUE(1), SVALUE(2))
         X = X + DELX
```

```
         enddo ! I
         print'(/a)',
        * '     X    YFIT  YFIT''  YFIT''''   '
         X = XI(1)
         do I=0,31                 !  Using Fortran 90 "DO" syntax.
            call DSVALA(KORDER, NCOEF, TKNOTS, 2, BDIF, X, SVALUE)
            IMAGE = ' '
            call DPRPL(SVALUE(0), '*', IMAGE, 49, SMIN, SMAX, .false.)
            call DPRPL(SVALUE(1), '1', IMAGE, 49, SMIN, SMAX, .false.)
            call DPRPL(SVALUE(2), '2', IMAGE, 49, SMIN, SMAX, .false.)
            print'(1x,f6.3,f7.3,f7.3,f7.3,1x,a)',X, SVALUE, IMAGE
            X = X + DELX
         enddo ! I
   c                          Compute and print residuals.
         print'(/a//a,a)',' Residuals at the data points:',
        *   '    I   XI(I)   YI(I)    YFIT   YFIT-YI(I)',
        *   '           YFIT-YI(I)'
         do I = 1, NDATA
            YFIT = DSVAL(KORDER, NCOEF, TKNOTS, BCOEF, XI(I), 0)
            call DPRPL(YFIT-YI(I), '*', IMAGE, 39, -0.18d0, 0.18d0, .true.)
            print'(1x,i4,f8.3,f8.3, f8.3, f10.3,1x,a)',
        *      I, XI(I), YI(I), YFIT, YFIT-YI(I), IMAGE(1:39)
         enddo ! I
         end ! DRDSFITC
```

ODDSFITC

```
DRDSFITC..  Demo driver for DSFITC
Least-squares polynomial spline fit to data with constraints.

  I   kind   deriv   relop  active       X         Y
  1   1       0       ~       a       0.000     1.000
  2   1       0       ~       a       0.300     1.100
  3   1       0       ~       a       0.700     0.900
  4   1       0       ~       a       1.000     1.020
  5   1       0       ~       a       1.300     1.200
  6   1       0       ~       a       1.700     1.000
  7   1       0       ~       a       2.000     1.200
  8   1       0       ~       a       2.300     1.400
  9   1       0       ~       a       2.500     1.760
 10   1       0       ~       a       2.600     2.000
 11   1       0       ~       a       2.800     2.400
 12   1       0       ~       a       2.900     2.600
 13   1       0       ~       a       3.000     3.000
 14   1       0       ~       a       3.100     3.400
 15   1       0       ~       a       3.200     3.700
 16   1       0       ~       a       3.500     4.300
 17   1       0       ~       a       3.700     4.450
 18   1       0       ~       a       4.000     4.760
 19   1       0       ~       a       4.300     4.800
 20   1       0       ~       a       4.700     5.000
 21   1       0       ~       a       5.000     4.960
```

```
 22   1       0       ~       a       5.300     4.900
 23   1       0       ~       a       5.700     4.900
 24   1       0       ~       a       6.000     5.000
 25   1       0       =       a       0.000     1.000
 26   1       1       >       a       0.000     0.000
 27   1       2       >       a       0.000     0.000
 28   1       2       >       a       1.500     0.000
 29   1       2       >       a       2.500     0.000
 30   1       2       <       a       3.500     0.000
 31   1       2       <       a       4.500     0.000
 32   1       2       <       a       6.000     0.000
 33   1       1       >       a       6.000     0.000
 34   1       0       =       a       6.000     5.000
 35                           !

  KORDER =  4,  NCOEF =  9
  TKNOTS() =    0.00000   0.00000   0.00000   0.00000
                1.50000   2.50000   3.30000   4.00000
                4.70000   6.00000   6.00000   6.00000
                6.00000

After call to DSFITC:

  IERR5 =     0,   NEED1 =     41,   NEED2 =    843
  M1    =    10,   MFIT  =     24,   NS    =      8
  RNORM =     0.37206

  BCOEF() =     1.00000   1.01613   1.04300   1.07848
                4.07150   4.87706   4.92040   4.96864
                5.00000

    X    YFIT  YFIT'  YFIT''
0.000  1.000  0.032  0.000                    2     *
0.200  1.006  0.032  0.000                    2     *
0.400  1.013  0.032  0.000                    2     *
0.600  1.019  0.032  0.000                    2     *
0.800  1.026  0.032  0.000                    2     *
1.000  1.032  0.032  0.000                    2     *
1.200  1.039  0.032  0.000                    2     *
1.400  1.045  0.032  0.000                    2     *
1.600  1.052  0.052  0.395                    1 2   *
1.800  1.076  0.210  1.186                     1    *2
2.000  1.147  0.527  1.977                       1   *    2
2.200  1.297  1.001  2.768                          1 *       2
2.400  1.558  1.634  3.559                             *1          2
2.600  1.959  2.360  3.045                                * 1   2
2.800  2.480  2.787  1.225                              2       *1
3.000  3.049  2.850 -0.595                  2                     1*
3.200  3.595  2.549 -2.414     2                              1     *
3.400  4.047  1.943 -3.064 2                                 1           *
3.600  4.377  1.382 -2.543    2                               1               *
```

```
 3.800  4.606  0.926 -2.022       2                1                      *
 4.000  4.755  0.573 -1.500          2           1                        *
 4.200  4.842  0.317 -1.065             2       1                         *
 4.400  4.887  0.147 -0.629               2    1                           *
 4.600  4.907  0.065 -0.194                  21                            *
 4.800  4.919  0.059  0.022                   2                            *
 5.000  4.931  0.063  0.019                   2                            *
 5.200  4.944  0.066  0.015                   2                            *
 5.400  4.957  0.069  0.011                   2                            *
 5.600  4.971  0.071  0.007                   2                            *
 5.800  4.986  0.072  0.004                   2                            *
 6.000  5.000  0.072  0.000                   2                            *
 6.200  5.014  0.072 -0.004                   2                            *
```

Residuals at the data points:

```
   I   XI(I)   YI(I)    YFIT   YFIT-YI(I)           YFIT-YI(I)
   1   0.000   1.000   1.000     0.000                  *
   2   0.300   1.100   1.010    -0.090           *     0
   3   0.700   0.900   1.023     0.123                 0            *
   4   1.000   1.020   1.032     0.012                 0*
   5   1.300   1.200   1.042    -0.158   *            0
   6   1.700   1.000   1.060     0.060                 0     *
   7   2.000   1.200   1.147    -0.053              *  0
   8   2.300   1.400   1.412     0.012                 0*
   9   2.500   1.760   1.740    -0.020                * 0
  10   2.600   2.000   1.959    -0.041              *   0
  11   2.800   2.400   2.480     0.080                 0       *
  12   2.900   2.600   2.763     0.163                 0              *
  13   3.000   3.000   3.049     0.049                 0    *
  14   3.100   3.400   3.330    -0.070            *    0
  15   3.200   3.700   3.595    -0.105         *       0
  16   3.500   4.300   4.226    -0.074            *    0
  17   3.700   4.450   4.503     0.053                 0     *
  18   4.000   4.760   4.755    -0.005                *0
  19   4.300   4.800   4.869     0.069                 0      *
  20   4.700   5.000   4.913    -0.087            *    0
  21   5.000   4.960   4.931    -0.029                *  0
  22   5.300   4.900   4.950     0.050                 0    *
  23   5.700   4.900   4.978     0.078                 0       *
  24   6.000   5.000   5.000     0.000                  *
```

11.6 Low-level Subprograms for Operations on Splines Functions

A. Purpose

This chapter describes five subprograms for spline operations that are used by the subprograms of the preceding chapter. It is expected that one would only use these subprograms directly if one has needs more specialized than are covered by the higher level subprograms of the preceding chapter.

Subroutine DSVALA evaluates at an argument X the values of the derivatives, of orders 0 through NDERIV, of a spline function represented using the B-spline basis. DSVALA must be given a difference table of the coefficients of the spline function. Subroutine DSDIF is provided to compute this difference table. Once the difference table has been computed and saved, use of DSVALA is more economical than making NDERIV+1 calls to subprogram DSVAL of the preceding chapter if NDERIV > 0.

Subroutine DSFIND does a lookup in a knot array to find a knot subinterval of nonzero length containing a specified argument X, or the nearest such subinterval if extrapolation is needed.

Using a knot sequence regarded as defining a B-spline basis function of order KORDER, subroutine DSBASD computes the values at X of the KORDER B-spline basis functions (or a derivative of these functions as specified by IDERIV) that could be nonzero at X. Subprogram DSBASI computes the integral from X1 to X2 of each of the NCOEF basis functions. The output of these subprograms is needed in setting up the matrix for curve fitting or interpolation involving values, derivatives, or integrals of the fitted spline function.

B. Usage

B1. Usage of DSBASD for Evaluation of Basis Functions or Their Derivatives

Program Prototype, Double Precision

```
INTEGER            KORDER, LEFT, IDERIV
DOUBLE PRECISION   TKNOTS(≥ncoef+KORDER), X, BDERIV(≥KORDER)
```

Assign values to KORDER, LEFT, TKNOTS, X, and IDERIV.

```
CALL DSBASD(KORDER, LEFT, TKNOTS, X, IDERIV, BDERIV)
```

Computed quantities are returned in BDERIV().

Argument Definitions

KORDER [in] KORDER is both the order of the spline basis functions and the number of basis functions whose derivatives are to be evaluated.

LEFT [in] Identifies an interval of nonzero length [TKNOTS(LEFT), TKNOTS(LEFT+1)] which is the *reference interval* for the function evaluation. DSBASD will evaluate the IDERIVth derivative of the KORDER basis functions that could be nonzero on this interval. Require KORDER ≤ LEFT ≤ *ncoef*. Except when extrapolation is needed, LEFT should satisfy TKNOTS(LEFT) ≤ X < TKNOTS(LEFT+1). We recommend that the subroutine DSFIND be used to determine LEFT.

TKNOTS() [in] The knots, t_i, i = 1, ..., *ncoef*+KORDER, where *ncoef* denotes the total number of B-spline basis functions associated with this knot sequence. The *proper interpolation interval*, [*a*,*b*], associated with this knot sequence is given by *a* = TKNOTS(KORDER) and *b* = TKNOTS(*ncoef*+1). Require $t_i \le t_{i+1}$ for i = 1, ..., *ncoef*+KORDER-1; $t_i < t_{i+KORDER}$ for i =

1, ..., *ncoef*; $t_{KORDER+1} > t_{KORDER}$; $t_{ncoef} < t_{ncoef+1}$. The knots strictly between *a* and *b* are *internal* knots. They specify abcissae at which one polynomial piece ends and the next begins. Successive internal knots may have the same value. An abcissa appearing with multiplicity μ means the order of continuity of the spline at this abcissa will be at least KORDER-μ-1. The knots indexed ahead of t_{KORDER} can all be equal to *a*, and those indexed after $t_{ncoef+1}$ can all be equal to *b*.

X [in] Argument at which the IDERIVth derivative of basis functions are to be evaluated.

IDERIV [in] Order of derivative to be computed. IDERIV = 0 specifies function values. Require IDERIV ≥ 0. Values of derivatives of order ≥ KORDER will be zero.

BDERIV()[out] On return the values at X of the IDERIVth derivative of the basis functions indexed from LEFT+1-KORDER through LEFT will be stored in BDERIV(*i*), *i* = 1, ..., KORDER.

B2. Usage of DSBASI for Evaluation of an Integral of Basis Functions

Program Prototype, Double Precision

```
INTEGER           KORDER, NCOEF, J1, J2
DOUBLE PRECISION  TKNOTS(≥NCOEF+KORDER), X1, X2, BASI(≥NCOEF)
```

Assign values to KORDER, NCOEF, TKNOTS(), X1, X2, J1, and J2.

```
CALL DSBASI ( KORDER, NCOEF, TKNOTS, X1, X2, J1, J2, BASI)
```

Computed results are returned in J1, J2, and BASI().

Argument Definitions

KORDER [in] The the order of the spline basis functions.

NCOEF [in] The total number of B-spline basis functions associated with this knot sequence. Also the number of values to be returned in BASI().

TKNOTS() [in] The knots, t_i, *i* = 1, ..., NCOEF+KORDER. The *proper interpolation interval*, [*a*,*b*], associated with this knot sequence is given by *a* = TKNOTS(KORDER) and *b* = TKNOTS(NCOEF+1). Require $t_i \le t_{i+1}$ for *i* = 1, ..., NCOEF+KORDER-1; $t_i < t_{i+KORDER}$ for *i* = 1, ..., NCOEF; $t_{KORDER+1} > t_{KORDER}$; $t_{NCOEF} < t_{NCOEF+1}$. The knots strictly between *a* and *b* are *internal* knots. They specify abcissae at which one polynomial piece ends and the next begins. Successive internal knots may have the same value. An abcissa appearing with multiplicity μ means the order of continuity of the spline at this abcissa will be at least KORDER-μ-1. The knots indexed ahead of t_{KORDER} can all be equal to *a*, and those indexed after $t_{NCOEF+1}$ can all be equal to *b*.

X1, X2 [in] Integration is to be done from X1 to X2. Permit X1 < X2 or X1 ≥ X2. Generally X1 and X2 should each lie in [*a*,*b*], however extraploation will be used to return values when this is not the case.

J1, J2 [inout] On entry, J1 and J2 must contain integer values. If J1 is in [1, N], it will be used to start the lookup for X1. Otherwise the search will start with 1. Similarly for J2.

On return, J1 and J2 indicate the portion of the array BASI() that might be nonzero on return. BASI(*i*) might be nonzero if J1 ≤ *i* ≤ J2, and BASI(*i*) = 0 if *i* < J1 or *i* > J2.

BASI() [out] On return, BASI(*i*) will contain the value of the integral of the *i*th basis function over the range from X1 to X2, for *i* = 1, ..., NCOEF. J1 and J2 above indicate which elements might be nonzero.

B3. Usage of DSDIF to Compute the Difference Table Needed by DSVALA

Program Prototype, Double Precision

```
INTEGER            KORDER, NCOEF, NDERIV
DOUBLE PRECISION   TKNOTS(≥NCOEF+KORDER), BCOEF(≥NCOEF),
                   BDIF(≥NCOEF*(NDERIV+1))
```

Assign values to KORDER, NCOEF, TKNOTS(), BCOEF(), and NDERIV.

```
CALL DSDIF (KORDER, NCOEF, TKNOTS, BCOEF, NDERIV, BDIF)
```

Computed results are returned in BDIF().

Argument Definitions

KORDER [in] The the order of the spline basis functions.

NCOEF [in] The total number of B-spline basis functions associated with this knot sequence.

TKNOTS() [in] Same specifications as for DSBASI above.

BCOEF() [in] Array of NCOEF coefficients representing a spline function relative to a B-spline basis.

NDERIV [in] Highest order difference to be computed. Since the difference table BDIF() is intended for use by DSVALA, this should correspond to the largest order derivative one intends to compute using DSVALA.

BDIF() [out] Will contain a copy of BCOEF() plus differences through order NDERIV of this array of coefficients. Intended for use by DSVALA.

B4. Usage of DSFIND for Lookup in a Knot Sequence

Program Prototype, Double Precision

```
INTEGER            IX1, IX2, LEFT, MODE
DOUBLE PRECISION   XT(IX2+1), X
```

Assign values to XT(), IX1, IX2, LEFT, and X.

```
CALL DSFIND(XT, IX1, IX2, X, LEFT, MODE)
```

Results are returned in LEFT and MODE.

Argument Definitions

XT(), IX1, IX2 [in] XT() is the array in which the lookup will be done. DSFIND will only look at elements from XT(IX1) through XT(IX2). Require IX1 < IX2, XT(IX1) < XT(IX1+1), XT(IX2-1) < XT(IX2), and XT(*i*) ≤ XT(*i*+1) for *i* = IX1, ..., IX2-1.

If the lookup is in a knot array of length *korder+ncoef* associated with a B-spline basis, one would generally set IX1 = *korder* and IX2 = *ncoef*+1. If the lookup is in a knot array of length *npc*+1 associated with a power basis, one would generally set IX1 = 1 and IX2 = *npc*+1.

X [in] Value to be looked up in XT().

LEFT [inout] On entry LEFT must contain an integer value. If this value is in [IX1, IX2-1] the lookup will start with this value, otherwise the lookup starts with IX1 or IX2-1.

On return LEFT is the index of the left end of the *reference interval* X(LEFT), X(LEFT+1) for X. This will always be an interval of nonzero length. If X satifies X(IX1) ≤ X < X(IX2) then LEFT will satisfy X(LEFT) ≤ X < X(LEFT+1). Otherwise, if X < X(IX1), LEFT := IX1; or if X ≥ X(IX2), LEFT := IX2-1. The polynomial segment defined over this reference interval is intended to be used for function evaluation at X.

MODE [out] Indicator of the position of X relative to [XT(IX1), XT(IX2)]. Set to -1 if X is to the left of this interval, to 0 if X is in this closed interval, and to +1 if X is to the right of this interval.

B5. Usage of DSVALA for Evaluation a Sequence of Derivatives

Program Prototype, Double Precision

```
INTEGER           KORDER, NCOEF, NDERIV
DOUBLE PRECISION  TKNOTS(≥NCOEF+KORDER), BDIF(≥NCOEF*(NDERIV+1)),
                  X, SVALUE(≥NDERIV+1)
```

Assign values to KORDER, NCOEF, TKNOTS(), NDERIV, BDIF(), and X.

```
CALL DSVALA(KORDER, NCOEF, TKNOTS, NDERIV, BDIF, X, SVALUE)
```

Computed results are returned in SVALUE().

Argument Definitions

KORDER [in] The the order of the spline basis functions.

NCOEF [in] The total number of B-spline basis functions associated with this knot sequence.

TKNOTS() [in] Same specifications as for DSBASI above.

NDERIV [in] Highest order derivative to be evaluated. Values of derivatives of order ≥ KORDER will be zero.

BDIF() [in] A difference table of B-spline coefficients computed by DSDIF.

X [in] Argument at which values returned in SVALUE() are to be computed.

SVALUE() [out] On return, SVALUE(*i*+1) contains the value at X of the i^{th} derivative of the spline function *f* for *i* = 0, ..., NDERIV. The spline function *f* is defined by the parameters KORDER, NCOEF, TKNOTS(), and the coefficients whose difference table in in BDIF().

B6. Modifications for Single Precision

For single precision usage change the DOUBLE PRECISION statements to REAL and change the initial "D" in the subprogram names to "S".

C. Examples and Remarks

The program DRDSBASD and output listing ODDSBASD demonstrate the use of the subprograms of this chapter.

D. Functional Description

The subprograms of this chapter are covered in Section D of Chapter 11.5.

References

1. Carl de Boor, *A Practical Guide to Splines*, Springer-Verlag, 1978.
2. D. E. Amos, Sandia report SAND79-1825, Sandia Laboratory, Albuquerque, June, 1979.

E. Error Procedures and Restrictions

DSBASD, DSBASI, and DSVALA each contain an internal dimensioning parameter *kmax* = 20. It is an error if KORDER > *kmax* in DSBASI, DSBASI, or DSVALA. The condition IDERIV < 0 is an error in DSBASD.

In DSFIND, if the search reaches either of the intervals [XT(IX1), XT(IX1+1)] or [XT(IX2-1), XT(IX2)] and the interval is found to have nonpositive length, the error is reported.

In each of the above cases the error is reported to the library error message processing subroutines of Section 19.2 with a severity level of 2 that will, by default, cause execution of the program to stop.

Abcissae and weights for 2-point, 6-point, and 10-point Gaussian quadrature are stored to 40 decimal digits in DSBASI. With infinite precision abcissae and weights, these formulae would be exact for splines of KORDER up to 20.

F. Supporting Information

The source language is ANSI Fortran 77.

Program Unit and Entry Name	**External References**
DSBASD	IERM1, IERV1
DSBASI	DSBASD, DSFIND
DSDIF	none
DSFIND	DERV1, IERM1
DSVALA	DSBASD, DSFIND
SSBASD	IERM1, IERV1
SSBASI	SSBASD, SSFIND
SSDIF	none
SSFIND	SERV1, IERM1
SSVALA	SSBASD, SSFIND

DRDSBASD

```
      program DRDSBASD
c>> 1993-01-12 CLL @ JPL
c>> 1992-11-12 CLL @ JPL
c>> 1992-11-04 CLL @ JPL
c   Demo driver for DSBASD, DSBASI, DSDIF, DSFIND, DSVALA.
c     ------------------------------------------------------------------
c--   D version uses DRDSBASD, DSBASD, DSBASI, DSDIF, DSFIND, DSVALA
c--   S version uses DRSSBASD, SSBASD, SSBASI, SSDIF, SSFIND, SSVALA
```

```
c     ------------------------------------------------------------------
      integer I, IDERIV, J1, J2, KORDER, LEFT, MODE, NCOEF, NDERIV
      parameter(KORDER = 4, NCOEF = 6)
      parameter( IDERIV = 2, NDERIV = 3)
      double precision BASI(NCOEF), BCOEF(NCOEF), BDERIV(KORDER)
      double precision BDIF(NCOEF*(NDERIV+1)), BVALS(KORDER)
      double precision SVALUE(NDERIV+1)
      double precision TKNOTS(KORDER+NCOEF), X, X1, X2
      parameter( X = 0.4d0 , X1 = 0.1d0, X2 = 0.9d0)
      data TKNOTS / 4*0.0d0, 0.3d0, 0.8d0, 4*1.0d0 /
      data BCOEF / 0.1d0, 0.4d0, 0.2d0, -0.3d0, -0.5d0, -0.2d0 /
c     ------------------------------------------------------------------
      print'(a/a/3x,a,i2,a,i2/3x,a,10f5.1)',
     * ' DRDSBASD..  Demo driver for',
     * '             DSBASD, DSBASI, DSDIF, DSFIND, DSVALA.',
     * ' KORDER =',KORDER,',  NCOEF =',NCOEF,
     * ' TKNOTS() =',(TKNOTS(I),I=1,KORDER+NCOEF)
      print'(a,f5.1)',' Using DSFIND with X = ', X
      LEFT = 1
      call DSFIND(TKNOTS, KORDER, NCOEF+1, X, LEFT, MODE)
      print'(3x,a,i2,a,i2)',' LEFT = ',LEFT,',  MODE =',MODE
      print*,'Using DSBASD with IDERIV = 0'
      call DSBASD( KORDER, LEFT, TKNOTS, X, 0, BVALS)
      print'(3x,a,i2,a,i2/6x,4f12.6)',
     *   ' Values at X of basis functions indexed from ',
     *     LEFT+1-KORDER,' to ', LEFT, (BVALS(I),I=1,KORDER)
      print'(a,i2)',' Using DSBASD with IDERIV =',IDERIV
      call DSBASD(KORDER, LEFT, TKNOTS, X, IDERIV, BDERIV)
      print'(3x,a,i2,a,i2/6x,4f12.5)',
     *   ' Values at X of 2nd deriv of basis functions indexed from ',
     *     LEFT+1-KORDER,' to ', LEFT, (BDERIV(I),I=1,KORDER)
      print'(a,f5.1,a,f5.1)',
     *   ' Using DSBASI with X1 = ', X1,' and X2 = ',X2
      J1 = 1
      J2 = 1
      call DSBASI(KORDER, NCOEF, TKNOTS, X1, X2, J1, J2, BASI)
      print'(3x,a,i2,a,i2/3x,a/3x,6f11.6)',' J1 =',J1,',  J2 =',J2,
     * ' Integrals from X1 to X2 of basis functions:',
     * (BASI(I),I=1,NCOEF)
      print'(a,i2,a/3x,a,6f5.1)',
     * ' Using DSDIF and DSVALA with NDERIV =',NDERIV,' and',
     * ' BCOEF() = ', (BCOEF(I),I=1,NCOEF)
      call DSDIF(KORDER, NCOEF, TKNOTS, BCOEF, NDERIV, BDIF)
      call DSVALA(KORDER, NCOEF, TKNOTS, NDERIV, BDIF, X, SVALUE)
      print'(3x,a,i2,a/6x,4f11.6)',
     *   ' Values of derivs 0 through ',NDERIV,' at X: ',
     * (SVALUE(I),I=1,NDERIV+1)
      end
```

ODDSBASD

```
DRDSBASD..  Demo driver for
            DSBASD, DSBASI, DSDIF, DSFIND, DSVALA.
   KORDER = 4,  NCOEF = 6
   TKNOTS() =  0.0  0.0  0.0  0.0  0.3  0.8  1.0  1.0  1.0  1.0
Using DSFIND with X =   0.4
   LEFT =  5,  MODE = 0
Using DSBASD with IDERIV = 0
   Values at X of basis functions indexed from  2 to  5
         0.200000    0.542857    0.253061    0.004082
Using DSBASD with IDERIV = 2
   Values at X of 2nd deriv of basis functions indexed from  2 to  5
          7.50000   -11.78571     1.83673     2.44898
Using DSBASI with X1 =   0.1 and X2 =   0.9
   J1 = 1,  J2 = 6
   Integrals from X1 to X2 of basis functions:
     0.014815   0.163874   0.246236   0.244080   0.127870   0.003125
Using DSDIF and DSVALA with NDERIV = 3 and
   BCOEF() =   0.1  0.4  0.2 -0.3 -0.5 -0.2
   Values of derivs 0 through  3 at X:
        0.110612  -1.181633  -1.132653   7.423469
```